EFFICIENT AND EFFECTIVE ALGORITHMS FOR CONTROLLABLE FLUID SIMULATION AND MESH DEFORMATION

BY

LIN SHI

B.S., Fudan University, 1999
M.S., New York University, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

Abstract

During the past decade, due to rapidly growing computational power, complicated numerical algorithms have become popular in graphics community which greatly improved the quality of graphics imagery. On the other hand, to produce high-quality visual effects, the size of the data being processed has also increased drastically at the same time. Thus efficient algorithms are desired in many applications. This dissertation focuses on problems in controlling fluid simulations and mesh deformations. It has potential applications in film and game industries.

The first part of this dissertation includes efficient algorithms for controlling the motion of gas and liquid. The efficiency is achieved by using feedback forces to drive the fluid toward the target. Without expensive optimization procedures, the control algorithms produce desired and natural results that only incur an additional computational cost linear to the size of the problem. Controlling the motion of lightweight natural objects in a gaseous medium is also studied, using data-driven synthesis based on stylistic motion planning and nonlinear optimization based on smoothed particle hydrodynamics. In addition, we also introduce a simple algorithm to perform plausible fluid simulation over triangle meshes.

To my parents and my girlfriend.

Acknowledgments

Over the last five years I have had the privilege of working with a variety of people who have made my time at UIUC an enjoyable and intellectually stimulating experience. I would like to thank all the people who have helped me along the way and contributed to this dissertation.

I am especially grateful to my advisor Yizhou Yu for taking me as his student and offering me the research opportunity in his group. His broad knowledge and dynamic thinking have guided me throughout the process of research. In working with Yizhou, I have learned how to pursue research problems with intellectual rigor and how to critically evaluate my work. Yizhou has also been incredibly patient with me and providing me valuable suggestions on thesis writing as well as presentation skills. Without him I could not have finished this work.

I would like to thank other members of my thesis committee, John C. Hart, David Forsyth and Steven LaValle, who have spent their valuable time reading this work, attending the defense and providing precious suggestions. Also I would like to thank Luke Olson for his important comments on the mesh deformation project.

One of the great things about working on a variety of different topics is that I have gotten to work closely with a lot of people. I have learned a lot from these collaborations and I would like to thank everyone I have worked with: Christopher Wojtan, Stephen Chenney, Nathan Bell, Wei-Wen Feng, Alan Perez-Rathke, Qing Wu and Hongcheng Wang.

Finally, I would like to dedicate this achievement to my family and my girlfriend. They have always believed in me, a priceless motivation which I truly appreciate. I hope that I have not disappointed them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Part I : Physically Plausible Fluid Simulation And Control

Fluids play an essential role in many beautiful natural phenomena. Because of that, exploring practical and plausible algorithms to simulate fluid motion has attracted a significant amount of research work in graphics community.

Equally important as simulating fluids is how to control them. Such effects appear in many practical applications, such as voxel water horses emerging from a flooding river [8], and the mummy manifests itself from sand. Early methods, such as vector fields [9] and embedded controllers [10], apply simple control over the motion of the fluids. In [11, 12], animator-designed "fake" space curves and surfaces can be applied to control the motion and structures of fluids. The tangents of the curves or the normals of the surfaces indicate the directions of motion. However, none of these attempts allows the user to enforce higher-level objectives, such as matching the shape of a still or moving object.

A technique to make fluids match regular shapes was first reported in [13], which involves carefully designed force fields and a modified diffusion equation for smoke gathering. A similar approach that can produce results with improved visual quality was also taken by [14]. Although being able to control the smoke density, the smoke gathering term in these techniques tend to generate "ghosting" effects, which mean that the target shape looks like simply emerging from an amorphous cloud of smoke.

Another class of methods to attach this problem is to use optimization technique. For smoke simulation the problem can be cast as matching dynamically evolved fluid density with the specified density distributions at a set of keyframes. The difficult part is to compute the derivatives of the simulation. An elegant method is introduced in [15]. The derivatives of the velocity field is dynamically simulated in the same framework for simulating the velocity field itself. However, since the derivatives of the velocity field with respect to each control parameter need to be computed throughout a portion of an animation sequence, this approach is computationally expensive. The adjoint method was adopted in [16] to significantly improve the efficiency of these derivative evaluations for each iteration. Since it is a gradient-based optimization, a number of

iterations are still necessary before convergence and there is also the peril of local minima.

An alternative to constrained simulation is fluid editing, or constructive methods for defining flows. Pidgin et al. [17] extract an Advected Radial Basis Function(ARBF) model from Eulerian simulations, which a user can control to a certain extent by editing the path-lines of the particles and maintaining coherence by enforcing spatial-temporal constraints. To simulate and control breaking waves, Mihalef et al. [18] introduced the Slice Method, in which a library of 2D breaking waves is used to integrate 3D shapes while the 3D geometry is being controlled. The subsequent dynamics is then computed with the aid of a 3D Navier-Stokes solver. Their algorithm is efficient and physical, but the trade-off is a lack of precise control over the final output.

The research in the first part of the thesis tries to solve these problems. It is essentially the balance between the effectiveness of control and the preservation of fluid appearance. Thus for smoke we use a shape transformation to connect an initial smoke region and the target shape both of which are represented as implicit functions, and impose carefully designed velocity constraints derived from a shape matching functional. This method is noniterative, and therefore, less expensive. The results maintain interesting dynamic fluid appearances around the target shapes for an arbitrarily long time. Since it does not involve extensive usage of source and sink, no "ghosting effects" are introduced. In addition, due to the distance transforms adopted, our method is less likely to be trapped in local minima. We then extend our work to controlling liquids to follow rapidly changing targets. A new control system is designed based on its unique properties. It consists of a shape feedback force and a derivative force which, in control terminology, roughly corresponds to proportional control and derivative control respectively. And the role of integral control is replaced with an adaptive geometric potential.

Besides the above applications, controllable lightweight objects are also widely used in synthetic animation and video production. The problem can be formulated as animating lightweight objects in a wind field, where the objects do not influence the air. Early work by Wejchert and Haumann [19] modeled the wind field as a linear combination of basis fields and advected leaves in the flow. Chenney [20] used a tile-based approach to define the flow which in turn is used to advect leaves. Both of these approaches offer indirect control of the object motion through control of the flow. Wei et al. [21] simulated the motion and deformation of lightweight objects in a wind generated by the Lattice-Boltzmann Model, but did not consider control. These algorithms are best suited for synthesizing and editing velocity fields instead of providing direct control over individual trajectories traced from specific locations, and the resulting fields may have visual artifacts because they are not physically motivated. We frame the problem as one of object motion synthesis in the presence of constraints imposed by the fluid. Inspired by the seminal work on spacetime

constraints [22], we formulate this problem as a large-scale optimization. Directly solving this large-scale problem using an off-the-shelf optimization toolbox would be infeasible since the number of parameters necessary for realistically describing fluid motion is prohibitively large. Therefore, we have devised a number of novel and effective methods to make this problem tractable. Data-driven synthesis is exploited to generate an initial trajectory for each object which may have certain desirable features required by the user. The optimization then makes the initial trajectories more physically plausible. The user is thus able to guide the optimization toward a specific local minima by biasing the starting point. During optimization, vortices are identified and recovered in a separate pass to improve efficiency. We ignore rotational components of the objects in the optimization and refine them in a separate post-processing stage to produce visually pleasing results.

Recently, fluid simulation has been generalized to Catmull-Clark subdivision surfaces [23]. This method does simulations in the surface parametrization space and alleviates parametrization distortions by incorporating the metric tensor. Though the amount of distortion was reduced significantly, it still exists and is noticeable. In addition to distortions, it is also hard to simulate incompressible flows in the parametrization space by enforcing the zero divergence property. Thus at the end of the first part, we propose to simulate directly on triangle meshes and thus eliminates parametrization distortions. It can enforce incompressibility on closed surfaces by utilizing a discrete vector field decomposition algorithm. It also includes effective implementations of semi-Lagrangian tracing and velocity interpolation schemes. Our implementation can produce convincing fluid motion on surfaces and has interactive performance for meshes with tens of thousands of faces.

1.2 Part II : Mesh Deformation Algorithms

Surface-based mesh editing has received much attention recently due to its capability to produce visually appealing results while at the same time making the underlying numerical computation transparent to the user. The user only needs to specify the goals, which can be handle positions, key frames or silhouettes, and the editing system automatically solves a sparse system of equations to satisfy these constraints. Nevertheless, a serious problem that still hampers the deployment of this type of techniques is their scalability. When meshes become large and complex, the performance of the numerical solver becomes the bottleneck of the entire system. While solutions to cope with this problem exist – including small regions of interest (ROI), precomputed matrix factorizations, and precomputed deformation bases – they restrict the scope of editing operations.

As another type of constrained animation problems, there is similarity between this and the controllable fluid simulation problem. The sparse linear systems developed here are usually for optimization purposes under a least-squares sense, and in most of the cases are Laplacian systems similar to the one solved during the projection step of fluid simulations. However, this system is now defined on an unstructured mesh or graph which renders traditional iterative solvers inefficient and poorly scalable. Besides, solving the system is the most time consuming part of the algorithm and requires higher accuracy than fluid simulation and control algorithms. Thus, the performance of preconditioned conjugate gradient, which is a popular solver used previously, is not acceptable any more. On the other hand, the factorization-based direct solvers widely used in mesh deformation algorithms come with a precomputation stage. This computationally intensive part does not scale well and consumes many CPU and memory resources, thus limits its application to relatively small sized problems and simpler scenarios where reutilizing the factorization is desired.

The multigrid method on the other hand has the potential to solve large-scale sparse systems efficiently without a significant setup time. While the user most often only needs to pick up a handle to manipulate a mesh, it is sometimes necessary to define and manipulate new handles that have not been preprocessed. Furthermore, in a general mesh editing environment, mesh deformation needs to be mixed with other mesh editing operations, such as remeshing and merging. This is evidenced by routine practice in game development where large-scale meshes are edited first before simplified to a size suitable for real-time rendering. Some of these editing operations result in an altered system of linear equations that need to be solved on the fly. When the mesh is large, a fast multigrid algorithm can solve the altered linear system in a less stressful way than the factorization stage in direct solvers.

In the second part, we introduce a fast multigrid technique tailored for mesh deformation to support the aforementioned scenario. Although the multigrid method has become a popular choice for large-scale mesh processing [24, 25], there are still a number of challenges we need to overcome to achieve acceptable interactive performance. First, recent mesh deformation techniques often have two passes with the first pass solving for local frames and the second pass solving for vertex coordinates. How can we effectively reformulate these passes so that they become more compatible with multigrid solvers? Second, the multigrid method requires a hierarchical structure and must accommodate user-provided deformation constraints within this hierarchy. How can we properly handle these aspects? Third, when moving between grid levels the multigrid method applies a pair of prolongation and restriction operators. How should we design such operators to speed convergence?

We have developed effective techniques to overcome these challenges. First, we revise the formulations for mesh deformation so that they can be adequately solved by local relaxations. In particular, we analyti-

cally obtain a closed-form formulation for the optimization of vertex coordinates, thereby avoiding expensive sparse matrix multiplications. Second, we quickly create a hierarchy using a simple graph coarsening technique that ignores the initial mesh structure. Boundary conditions are not explicitly considered during the construction of the hierarchy. Instead, they are incorporated algebraically in the equations and in the prolongation/restriction operators. Most importantly, we develop a novel technique that automatically obtains prolongation and restriction operators using a weighted graph perspective. These operators better maintain the consistency of equations among different levels, and thus significantly improve the convergence rate. As a result, our algorithm can outperform existing multigrid solvers and the factorization stage of direct solvers. We demonstrate the advantages and utility of these features in complex mesh editing examples.

Chapter 2

Fluid Simulations

2.1 Navier Stokes Equation

The dynamics of a compressible fluid can be modeled using the following Navier-Stokes equations:

$$\frac{\partial \rho_f}{\partial t} = -\nabla \cdot (\rho \mathbf{u}) \tag{2.1}$$

$$\rho_f \frac{\partial \mathbf{u}}{\partial t} = -\rho_f (\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \frac{\nu}{3}\mathbf{H}(\mathbf{u}) + \rho_f \mathbf{f} - \nabla P \tag{2.2}$$

where ρ_f is the density of the fluid which is different from the smoke density, $\mathbf{u}$ represents the velocity field, P is the pressure term, $\mathbf{f}$ is an external force field, and ν is the kinematic viscosity of the fluid. The x component of $\mathbf{H}(\mathbf{u})$ is $\nabla \cdot (\frac{\partial \mathbf{u}}{\partial x})$, and the y and z components are defined similarly. The first equation arises from the conservation of mass, but the volume of the fluid is compressible with increasing density for a decreasing volume. The second equation is for the conservation of momentum; the first term on the right hand side is the convective term; the second and third terms model accelerations due to viscous forces; the last two terms model accelerations due to external forces and forces arising from the pressure gradient.

These equations can effectively model high-speed velocity fields such as shock waves generated by explosions [26]. However, a strict time step condition is necessary for stable numerical solutions. For liquids and low-speed gaseous phenomena, the compressibility effects are negligible, and conservation of mass becomes equivalent to conservation of volume. The assumption of incompressibility leads to the following equations which make more efficient numerical methods possible.

$$\nabla \cdot \mathbf{u} = 0 \tag{2.3}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho_f}\nabla P + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{2.4}$$

where the first equation means the fluid is volume-preserving, and the second one is similar to the corresponding equation in the aforementioned model for compressible fluids.

2.2 Simulating Smoke with Stable Fluids

If we focus on gases, the effects of viscosity are also negligible. Therefore, the diffusion terms in the above models can be left out. Simulating gaseous phenomena thus reduces to solving the following incompressible and inviscid Euler equations [27, 28, 29]:

$$\nabla \cdot \mathbf{u} = 0 \tag{2.5}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla P + \mathbf{f} \tag{2.6}$$

where the factor $\frac{1}{\rho_f}$ in (2.4) has been integrated into the "pressure" P which is used to guarantee zero divergence, but does not represent actual pressure any more.

The temperature and density of the smoke are assumed to be controlled by the following equations

$$\frac{\partial T}{\partial t} = -\mathbf{u} \cdot \bigtriangledown T \tag{2.7}$$

$$\frac{\partial \rho}{\partial t} = -(\mathbf{u} \cdot \bigtriangledown)\rho + S_\rho \tag{2.8}$$

where the first terms on the right hand side indicate the advection of these scalar fields along the velocity field of the gas, and S_ρ is a source term with negative values for a sink.

A numerical implementation solving the above incompressible scheme consists of the following basic steps [29]:

1. Compute an intermediate fluid velocity field, $\{\mathbf{u}^*\}$, from (2.6) ignoring the pressure term by first adding external force times the time step, and then solving the advection part $(\mathbf{u} \cdot \nabla)\mathbf{u}$ by using semi-Lagrangian tracing [30];

2. Obtain the pressure P by solving the Poisson equation,

$$\nabla^2 P = \frac{1}{\triangle t} \nabla \cdot \mathbf{u}^*; \tag{2.9}$$

   where $\triangle t$ is the size of the time step;

3. Obtain the divergence free component of $\{\mathbf{u}^*\}$ by subtracting the gradient of P from the intermediate velocity,

$$\mathbf{u} = \mathbf{u}^* - \triangle t \nabla P. \tag{2.10}$$

4. Advect the density field ρ using semi-Lagrangian tracing again.

The advection part is solved using a semi-Lagrangian scheme [30] instead of finite difference. That is, from the currently considered voxel, trace a path along the reversed velocity direction, and the path ends when the accumulated elapsed time reaches $\triangle t$. The velocity at the end of the path is transferred to the currently considered voxel. The path tracing part is carried out by further dividing the time step into subintervals and tracing the velocity field at these subintervals.

There are two typical boundary conditions for the Poisson equation, the Neumann boundary condition and Dirichlet boundary condition. The Neumann boundary condition imposes $\frac{\partial p}{\partial \mathbf{n}} = 0$ at a boundary point with normal $\mathbf{n}$. The Dirichlet boundary condition specifies the pressure at boundary points directly instead of its normal derivative. Preconditioned conjugate gradient (PCG) with Incomplete Cholesky Factorization (ICF) [31] is a natural choice to solve this sparse linear system.

To transport smoke density, the advection term of Eq. (2.8) uses the gas velocity field to move smoke. It is implemented by using the semi-Lagrangian scheme again. This time, it is the smoke density, instead of the velocity itself, that is being transferred.

2.3 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics represents a fluid as a collection of moving elements, particles, with local fluid characteristics. Each particle has a mass, position and velocity and is influenced by forces such as gravity. In addition, an SPH particle also has local fluid characteristics such as density and pressure. The idea behind SPH is the determination of characteristics of fluid by interpolating from the set of unorganized particles. The interpolation is performed as a weighted sum over particles within a local region defined by a smoothing length h. The weighting scheme is defined by a smoothing kernel $w(r, h)$ which can be a Gaussian or a polynomial with a finite support. The smoothing length defines the scale of the support. As an example, the smoothed estimate of the density at particle i can be formulated as $\rho_i = \sum_j m_j w_{ij}$, where $w_{ij} = w(\|\mathbf{x}_i - \mathbf{x}_j\|, h)$.

Computing the gradient of an interpolated property is done using the gradient of the smoothing kernel, giving a smoothed estimate of the gradient of the property. Thus, the SPH versions of the Lagrangian equations of motion,

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \mathbf{v} \tag{2.11}$$

$$\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P \tag{2.12}$$

$$\frac{d\tau}{dt} = -\frac{P}{\rho}\nabla \cdot \mathbf{v} \tag{2.13}$$

m	Mass	t	Time
ρ	Density	$\mathbf{x}$	Position
P	Pressure	$\mathbf{v}$	Velocity
τ	Thermal energy	$\mathbf{q}$	State
E	Cost function	$\triangle\mathbf{v}$	XSPH variant
		$\mathbf{F}$	Force

Table 2.1: Frequently used notations.

can be written as

$$\begin{aligned}
\frac{d\rho_i}{dt} &= \sum_j m_j \mathbf{v_{ij}} \cdot \nabla_i w_{ij} \\
\frac{d\mathbf{v_i}}{dt} &= -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ij} \right) \nabla_i w_{ij} + \mathbf{F_i} \\
\frac{d\tau_i}{dt} &= \frac{1}{2} \sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} + \Pi_{ij} \right) \mathbf{v_{ij}} \cdot \nabla_i w_{ij}
\end{aligned} \tag{2.14}$$

where $\mathbf{v_{ij}} = \mathbf{v_j} - \mathbf{v_i}$, Π_{ij} is an artificial viscosity added to handle shocks, and $\mathbf{F_i}$ is the external force on particle i. The interpretation of other notations are given in Table 2.1. These three equations maintain the conservation of mass, momentum and thermal energy, respectively. Particle interactions are implicitly handled by the smoothing kernel.

In addition to these equations, an *equation of state* must be used to fully describe the behavior of fluid. This equation defines a functional relationship between temperature, density and pressure. An example of the equation for an ideal gas is: $P = (\gamma - 1)\rho\tau$, where γ is a parameter that depends on the gas being simulated.

To put these equations in perspective, we summarize the steps a typical SPH simulation goes through. Particles are initialized to have an initial position, velocity, mass and energy and the system is evolved as follows:

```
Update particle densities
Update particle pressures using the equation of state
while (time < end_of_time){
  for (all particles){
    Calculate acceleration due to pressure gradient
    Calculate rate of change of thermal energy
  }
  for (all particles){
    Update position
    Update velocity
    Update thermal energy
  }

  Update particle densities
  Update particle pressures
  Calculate new time step
  time += new_timestep
}
```

Appropriate numerical schemes for SPH include the leapfrog algorithm and Runge-Kutta methods. Because of the finite smoothing length, the number of pairwise interactions is actually proportional to the number of particles, making SPH simulations efficient if a spatial data structure is used.

In order to simulate incompressible or nearly incompressible (such as a stiff gas) fluids, [32, 33] adapted the original SPH. To prevent inter-penetrations, a velocity correction, called the XSPH variant is added when the particle positions are updated. XSPH variant is defined to be

$$\triangle \mathbf{v}_i = \varepsilon \sum_j \frac{m_j \mathbf{v}_{ji}}{\bar{\rho}_{ij}} w_{ij} \tag{2.15}$$

where $\bar{\rho}_{ij} = (\rho_i + \rho_j)/2$. A particle position is updated as follows:

$$\mathbf{x}_i' = \mathbf{x}_i + \triangle t(\mathbf{v}_i + \triangle \mathbf{v}_i). \tag{2.16}$$

In addition, a different equation of state, that keeps compressibility below a few percent, is also adopted:

$$P_i = P_0 \left[\left(\frac{\rho_i}{\rho_0} \right)^{\gamma} - 1 \right] \tag{2.17}$$

where P_0 is the reference pressure.

2.4 Simulating Water with Level Set

Instead of nice details throughout the whole space as smoke, water featured in a clear interface it shared with the air. The water is described as a signed distance function ϕ and the interface as its 0 level set, $\{\mathbf{x} | \phi(\mathbf{x}) = 0\}$.

Comparing to water, the density of air is negligible, same as its influence on the motion of water. Thus we only need to model the velocity field of the water itself. The algorithm used to simulate water extended that in 2.2 with minor changes incorporating the effect of the water interface.

Advection of the velocity field only need to be performed inside the fluid, and velocity extrapolation is used to obtain a smooth, divergence free velocity field in the "air" region near the interface. To calculate the extrapolated velocity, $\vec{u}_{ext} = (u_{ext}, v_{ext}, w_{ext})$, away from the interface, we solve to steady state

$$\frac{\partial u_{ext}}{\partial \tau} + \nabla \phi \cdot \nabla u_{ext} = 0 \tag{2.18}$$

$$\frac{\partial v_{ext}}{\partial \tau} + \nabla \phi \cdot \nabla v_{ext} = 0 \tag{2.19}$$

$$\frac{\partial w_{ext}}{\partial \tau} + \nabla \phi \cdot \nabla w_{ext} = 0 \tag{2.20}$$

where τ is a fictitious time. The above equations are solved in band of 5 grid cells thick near the interface. Since the solution should be constant normal to the interface, a fast marching technique can be used to update $\vec{u}_{ext}$. The pressure of air voxels are fixed as 0 and the poisson equation only need to be solved in voxels containing water.

The level set function is advected with the level set equation

$$\phi_t + \mathbf{u} \cdot \nabla \phi = 0 \tag{2.21}$$

Reinitialization algorithms maintain the signed distance property by solving to steady state(as fictitious time $\tau \to \infty$ the equation

$$\phi_\tau + \frac{\phi_0}{\sqrt{\phi_0^2 + (\triangle x)^2}}(|\nabla \phi| - 1) = 0 \tag{2.22}$$

In practice, the above equation is performed for 10 iterations in a narrow band(10 grid cells) about the interface. And the rest of the space is filled with the result of fast marching method.

The velocity field and level set function are updated using different but related time step. The CFL condition

$$\triangle t(\frac{|u|_{max}}{\triangle x} + \frac{|v|_{max}}{\triangle y} + \frac{|w|_{max}}{\triangle z}) \leq 1 \tag{2.23}$$

asserts that the numerical waves should propagate at least as fast as the physical waves. In practice the level set function is evolved according to

$$\triangle t_{levelset} = .5 \min(\frac{\triangle x}{|u|_{max}}, \frac{\triangle y}{|v|_{max}}, \frac{\triangle z}{|w|_{max}}) \tag{2.24}$$

The liquid velocity field, $\vec{u}$, is updated at a rate given by

$$\triangle t_{liquid} = 4.9 \triangle_{levelset}$$

Between the liquid velocity updates, the level set function are evolved forward in time according the most recently computed velocity field.

Chapter 3

Controllable Smoke Animation with Guiding Objects

Amorphous but elegantly moving matters, such as clouds, fog and smoke, usually give people plenty of space for imagination. We would be excited when a cloud in the sunset sky assumes the approximate shape of an animal or some other real object. It is indeed an exhilarating event because of its rareness. For the same reason, ghosts and deities are usually described to manifest themselves from smoke or clouds. Even a fairy tale has the following scene: as Aladdin rubbed the lamp to try to get a better look, the lamp came to life; the lamp launched a long, blue stream upward; the blue smoke rose toward the ceiling, and finally became an enormous, blue genie!

We would like to develop techniques for digitally reproducing similar effects. Such techniques have many applications in the entertainment industry especially in advertising and film making. In some of the recent movies, there have been voxel water horses emerging from a flooding river [8], and the mummy manifests itself from sand. Our goal in this work is to introduce methods that produce physically plausible motion for a gas phenomenon, which at the same time, assumes a recognizable static or dynamic shape. In the rest of the work, we choose *smoke* as a representative of such gas phenomena. Nevertheless, the approach introduced here is not only limited to smoke.

Our goal in this work has the following implications:

- the motion during two-way transitions between irregular smoke regions and regular object shapes should be natural and have realistic smoke appearances;
- the global shape of the smoke should be able to approximate a static or moving object for an arbitrarily long period of time while maintaining its characteristic local structure and motion;
- When smoke objects interact with each other or with the environment, the objects should exhibit the properties of smoke so that a strong wind or other regular objects can easily destroy the shape of such objects.

An example with a smoke horse is shown in Fig. 3.1(a) to illustrate this goal.

Smoke consists of a collection of light-scattering tiny particles floating in the air. Creating the above dramatic effects is challenging since the smoke density in a fluid medium always tends to drift from a nonuniform distribution to a uniform one. Solving the proposed problem requires the maximum level of control of this process while maintaining a believable appearance of smoke. When there is a conflict between controllability and physics rules, we choose to relax the physics rules since the desired effects can largely be considered as a supernatural phenomenon. Our goal in this work is consistent with one of the general objectives of graphics research: the development of techniques that allow easy user-level control of the modeling and animation processes.

This work presents an effective solution to the proposed problem. Our solution involves implicit functions (level sets) defined for both smoke-object transitions and object motion. Since they are functions of both space and time, these level sets represent both the shape of the target objects as well as their evolution over time. These implicit functions serve as the underlying "storyboard" guiding the motion of the smoke. Thus, the problem becomes how to impose constraints on the motion of the smoke so that the smoke density distribution approximately matches these evolving level sets while the realistic appearance of smoke is being maintained. This is actually a control problem that can be solved by a dynamic feedback process. The basic idea of our solution lies in the use of artificial feedback forces on the smoke so that subtle changes in the movement of the smoke reduces the shape discrepancy between the smoke and the target object. Such feedback forces are actually realized by velocity adjustments and constraints. They do not exist in the real world, and need to be carefully orchestrated to achieve the desired effects.

Major contributions of this work include an overall framework for solving the proposed problem, an automatic scheme for target object matching based on velocity constraints imposed on the motion of the smoke, and an empirical compressible fluid model for effectively integrating constraints into the velocity field. These velocity constraints are derived from a shape matching functional. Simple but effective methods for smoke objects to interact with each other or the environment are also developed.

3.1 Overview

In our method, we define an implicit function for the target object and try to drive an irregularly shaped smoke region so that a specific isosurface of the density distribution of the smoke closely matches the boundary (zero) level set of the target object. The implicit function for the target object is named the *guiding implicit function* and is denoted as $D(\mathbf{x}, t)$ where $\mathbf{x}$ represents a point in a 2D or 3D space, and the extra variable t means the target shape may move or deform over time. The density distribution of

the simulated smoke is denoted as $\rho(\mathbf{x}, t)$. We use dynamic force feedback to actively influence the smoke isosurface so that the resulting dynamic surface represented by $\Gamma_\rho = \{x|\rho(\mathbf{x}, t) - \tau = 0\}$ approximately matches the zero level set of the target object, $\Gamma_D = \{x|D(\mathbf{x}, t) = 0\}$ where τ is a threshold used for defining a boundary isosurface for the smoke region which is assumed to have internal densities higher than τ. Note that $\rho(\mathbf{x}, t) - \tau = 0$ is actually an implicit function for the smoke region. At every time step, given the shape discrepancy between the two isosurfaces, feedback forces are applied to the smoke boundary to reduce the amount of discrepancy. Since forces and accelerations are connected through the Newton's law, in practice, such feedback forces are actually realized by velocity adjustments.

There are two essential stages in an animation where smoke evolves into an object. The first stage involves a shape transition between the smoke region and the target object. In the second stage, the transformed smoke region needs to keep track of the object's own nonrigid deformation or rigid-body motion. These two stages can be treated in a unified framework. The initial smoke region obtained by thresholding can be considered as an irregular object. 3D shape morphing techniques can be applied to generate a morph sequence between the initial smoke shape and the target object shape (Fig. 3.1(b)). As a result, we can obtain an intermediate shape at any instance during the whole transition period. Thus, the shape transition between the smoke and the target object can be viewed as a nonrigid shape deformation, and any method designed for the second stage can be applied as well to this stage where the morph sequence is used to generate the guiding implicit function which the smoke tries to match. Therefore, we only need to develop an approach for the smoke to track object motion. A static object is a special case for this problem.

Nevertheless, tracking object motion using the smoke is by no means trivial. When there is temporal coherence and the frame-to-frame motion of the target object is small, it is possible to obtain velocity constraints on the boundary isosurface of the smoke by exploiting the local gradient flow of the boundary. If a physical simulation of the smoke satisfies these boundary velocity constraints, the difference between the two boundary isosurfaces will be decreased. However, when there is fast frame-to-frame motion, the target object from two consecutive frames might have a huge gap in position or orientation. Local gradient structure of the implicit functions cannot guarantee efficient tracking results any more. When this happens, we use the target object to transport the smoke inside from its location in the previous frame to its location in the current frame as if the target object is a container with a hard boundary. The hard boundary disappears once the smoke is in position again. Note that this step may not be physically realistic, and is specially designed to achieve the desired phenomenon.

The overall solution to the proposed problem has the following components:

- The animator needs to pick a target object and chooses a smoke region from a simulation. A 3D morph

sequence is created between the initial smoke shape and the target object. The target object for each frame is voxelized into a discrete implicit function.

- For each frame, the system determines whether large motion has occurred by checking the amount of overlap between the two underlying object shapes at the previous and current frames. The strategy for smoke transportation is executed if the amount of overlap falls below a predefined threshold. Small motion tracking is always performed no matter whether large motion occurred or not.

- During each iteration, the system also simulates potential interactions among multiple smoke objects and interactions between a smoke object and its environment. The shape of a smoke region may be destroyed during such interactions, and a new morph sequence can be generated between the remaining smoke and the target object.

3.2 Guiding Objects

The input to our system includes the target objects and their motion. When the objects are not static, the configuration of the objects needs to be specified at every frame. The type of motion may include simple rigid-body motion, more complicated articulated body motion, or even nonrigid deformations. All target objects are internally represented as implicit functions. Our representation for the guiding implicit function $D(\mathbf{x}, t)$ is the signed distance function which is zero on the object boundary and positive at the interior. The signed distance function has the advantage to conveniently provide the shortest distance between any point on the smoke boundary and the boundary of the guiding implicit function. Therefore, input objects need to be converted to this representation even if they are already given as implicit functions.

Since smoke simulation will be performed on a volume grid, the smoke implicit function $\rho(\mathbf{x}, t) - \tau$ is always directly represented on this discrete grid in the form of a density value at each voxel. The boundary isosurface of this function is obtained by labeling the voxels containing a density value close to the threshold τ. The signed distance function of a guiding object at a specific frame is also represented on the same volume grid by first discretizing the object's original representation followed by a conversion to signed distance values.

As discussed at the beginning of this chapter, a part of the animation may involve a shape transition between the smoke region and the target object. An intermediate shape should be generated at each frame during this transition. This intermediate shape serves as the guiding object for that frame. And it needs to be represented as an implicit function as well. In practice, we apply 3D shape morphing and masking techniques (see Section 3.5.1) to generate a shape transition sequence which can produce such an intermediate shape at any intermediate frame. These intermediate shapes should be represented as or converted to signed

distance functions whenever they are needed.

3.3 Shape Matching

3.3.1 Velocity Constraints for Small Motion

Suppose at a certain time t_i during simulation, the guiding implicit function is $D(\mathbf{x}, t_i)$. We would like to evolve the smoke density at the previous step $\rho(\mathbf{x}, t_{i-1})$ so that the zero isosurface of the updated function $\rho(\mathbf{x}, t_i) - \tau$ approximately matches the boundary of the guiding object.

Since we are concerned with matching two isosurfaces, let us first look at a criterion for measuring shape discrepancy. If we represent an object as a point set, two objects A and B exactly match each other if and only if both sets $A - B$ and $B - A$ are empty, which is also equivalent to that the volumes of both $A - B$ and $B - A$ are zero. The summed volumes of these two sets indicate the level of discrepancy between two shapes. Mathematically, we need to use characteristic functions of the shapes and integrals to represent these two volumes. Let us define $\chi_D(\mathbf{x}, t_i) = 1$ if $D(\mathbf{x}, t_i) \geq 0$, and $\chi_D(\mathbf{x}, t_i) = 0$, otherwise; and define $\chi_\rho(\mathbf{x}, t_{i-1}) = 1$ if $\rho(\mathbf{x}, t_{i-1}) - \tau \geq 0$, and $\chi_\rho(\mathbf{x}, t_{i-1}) = 0$, otherwise. The level of discrepancy between the two zero isosurfaces can be measured by the following integral,

$$e_v = \int \chi_\rho(\mathbf{x}, t_{i-1})(1 - \chi_D(\mathbf{x}, t_i))d\mathbf{x} + \int \chi_D(\mathbf{x}, t_i)(1 - \chi_\rho(\mathbf{x}, t_{i-1}))d\mathbf{x} \tag{3.1}$$

where the global minimum is zero and can be reached when the two characteristic functions coincide. (3.1) can be simplified to $\int \chi_D d\mathbf{x} + \int \chi_\rho(1 - 2\chi_D)d\mathbf{x}$ where χ_ρ is the shape variable and χ_D is the fixed guiding shape for a specific frame since we would like $\rho(\mathbf{x}, t_{i-1})$ to approximate $D(\mathbf{x}, t_i)$. Therefore, reducing the shape discrepancy between the two is equivalent to minimizing the following functional,

$$\int \chi_\rho(1 - 2\chi_D)d\mathbf{x}. \tag{3.2}$$

According to calculus of variations [34], one can show that the first variation of the integral in (3.2) with respect to the smoke boundary surface is simply based on the second part of its integrand and the normal directions of the smoke boundary. Therefore, the negative variational gradient minimizing the functional in (3.2) with respect to the smoke boundary is as follows.

$$\left.\frac{\delta e_v}{\delta \Gamma_\rho}\right|_{\mathbf{x}=\mathbf{x_b}} = (1 - 2\chi_D) \left.\frac{\nabla\rho}{\|\nabla\rho\|}\right|_{\mathbf{x}=\mathbf{x_b}} \tag{3.3}$$

where Γ_ρ represents the smoke boundary and $\frac{\nabla\rho}{\|\nabla\rho\|}$ represents the unit inward normal at the smoke boundary. Detailed discussion on the derivation of this variational gradient is presented in Appendix A.

Consider a specific point $\mathbf{x_b}$ on the boundary of the smoke, the sign of $D(\mathbf{x_b}, t_i)$ indicates its location with respect to the guiding object. Note that the gradient of $\rho(\mathbf{x}, t_{i-1})$ at $\mathbf{x_b}$ points to the interior of the smoke and is perpendicular to the boundary. Under the small motion assumption, the smoke and guiding object should have overlap. If $\mathbf{x_b}$ is inside the guiding object, we need to move it slightly along the negative gradient direction; otherwise, move it along the positive gradient direction. (3.3) means iteratively perturbing all the smoke boundary points simultaneously in this way would gradually decrease the costs in (3.2) and (3.1). Since a new smoke boundary is formed by the relocated points after each iteration, the gradient of the new boundary should be used for moving points in the subsequent iteration. This scheme for functional minimization bears resemblance to gradient descent for regular function minimization. As gradient descent, this scheme cannot always converge to the global minimum if the initial shape of the unknown is not sufficiently close to the target shape. For example, if the smoke region does not have overlap with the guiding object, this scheme would gradually shrink the smoke region until it disappears and at the same time decrease the cost in (3.1) to be the volume of the guiding object which is actually a correct local minimum.

In the current context, we can modify the above iterative minimization scheme to avoid the local minimum for two separate shapes. When $D(\mathbf{x_b}, t_i)$ is negative, indicating $\mathbf{x_b}$ is outside the guiding object, it should be moved along a direction along which the directional derivative of $D(\mathbf{x}, t_i)$ is positive to bring it closer to the boundary of the guiding object. Since this condition must be satisfied either by the positive or by the negative gradient direction of $\rho(\mathbf{x}, t_{i-1})$, we only need to choose the right one of these two instead of always following the positive gradient. We still keep the original scheme when $\mathbf{x_b}$ is inside the guiding object.

Since we would like to realistically evolve the boundary of $\rho(\mathbf{x}, t_{i-1}) - \tau$ into the target shape, the dynamic evolution should follow the above iterative procedure to minimize the integral in (3.1) as well as follow the physics rules in smoke simulation as closely as possible. To achieve these goals, the smoke simulation should satisfy velocity constraints derived from the minimization procedure. Since (3.3) shows a first-order scheme, we should adopt its direction field but adjust its magnitude to improve stability.

A velocity $\mathbf{u}$ on the smoke boundary can be decomposed into a normal component $\mathbf{u_n}$ and a tangential component $\mathbf{u_t}$. Based on (3.3), the normal component $\mathbf{u_n}$ at a smoke boundary point $\mathbf{x_b}$ at time t_i is defined to be

$$\mathbf{u_n}(\mathbf{x_b}, t_i) = C_n \cdot \min(d_{max}, |D(\mathbf{x_b}, t_i)|) \cdot \frac{\nabla\rho}{||\nabla\rho||} \cdot \mathrm{msgn}(\mathbf{x_b}, t_i) \tag{3.4}$$

where C_n is a constant scaling parameter, $|D(\mathbf{x_b}, t_i)|$ is the magnitude of the signed distance function,

indicating how far away $\mathbf{x_b}$ is from the current boundary of the guiding object, d_{max} is a clamping upper bound for the distance, and $\text{msgn}(\mathbf{x_b}, t_i)$ adjusts the direction of the velocity vector according to our modified scheme. Specifically, $\text{msgn}(\mathbf{x_b}, t_i) = -1$, if $D(\mathbf{x_b}, t_i) > 0$; $\text{msgn}(\mathbf{x_b}, t_i) = \text{sgn}(\nabla D \cdot \nabla \rho)$, if $D(\mathbf{x_b}, t_i) \leq 0$. The incorporation of the term $\min(d_{max}, |D(\mathbf{x_b}, t_i)|)$ can alleviate overshooting when $\mathbf{x_b}$ is already close to the boundary of the guiding object. However, this term should only be present in the normal component when the smoke region already has some overlap with the guiding object. Otherwise, it would reduce the velocity of the smoke to zero when it touches the boundary of the guiding object, and keep the smoke from entering the object. Various situations for setting up normal velocity constraints are summarized in Fig. 3.2. Note that when the volume of the smoke differs from the object, these velocity constraints do not preserve mass. This is a tradeoff we need to make between physics and shape matching.

Constraints on the tangential component are also crucial. Although the tangential component does not directly affect the shape of the level sets, it does affect the surrounding velocity field. While a zero tangential component would make the smoke surface less alive, the numerical stability of the shape matching procedure may be compromised if it becomes overly large. The maximum allowable tangential component is actually dependent on the local geometry of the smoke surface. If the surface is flat, the tangential displacement can be large without destroying the original shape while a highly curved surface is certainly more vulnerable. Therefore, the magnitude of the tangential component should have an upper bound related to the surface curvature. Since we would like to follow physical simulation as faithfully as possible, we can simply clamp the physically generated tangential components against the upper bound when their magnitude becomes too large. Thus, the constrained tangential component is simply defined to be

$$\mathbf{u_t}(\mathbf{x_b}, t_i) = \min(||\mathbf{u_t^*}||, \frac{C_t}{K}) \frac{\mathbf{u_t^*}}{||\mathbf{u_t^*}||} \tag{3.5}$$

where $\mathbf{u_t^*}$ is the tangential component generated from a simulation, C_t is a constant parameter, and K is the surface curvature. In practice, we use mean curvature. A robust implementation of mean curvature on a volume grid can be found in [35].

Note that the normal and tangential velocity constraints are derived velocities for matching two shapes. They are different from those constraints defined in [29, 11] for fluid-object interaction and fluid control. Our guiding objects are invisible "ghost" objects and they do not directly interact with the smoke.

3.3.2 Velocity Constraints for Large Motion

When the frame-to-frame motion of the guiding object becomes excessively large, the two instances of the guiding object at two consecutive frames may have little overlap or no overlap at all. Although the matching scheme developed in the previous section can eventually converge, many iterations would be needed to actually reach convergence. For efficiency considerations, we directly transport the smoke from the location of the first object instance to the second. During this direct transportation, the guiding object is assumed to have hard boundaries and the part of the smoke that is already inside becomes trapped and moved together with the guiding object. Meanwhile, velocity constraints should still be imposed at the boundary and interior of the guiding object so that these constraints can bring along the surrounding region and generate turbulent flows to create a fluid appearance and "evidences" for the large motion. These constraints are not for transporting the smoke inside the guiding object. Suppose a point on the guiding object moved from $\mathbf{x_{i-1}}$ to $\mathbf{x_i}$ during two consecutive frames at time t_{i-1} and t_i, a velocity constraint

$$\mathbf{u_L} = \frac{\mathbf{x_i} - \mathbf{x_{i-1}}}{t_i - t_{i-1}}$$

should be imposed at $\mathbf{x_{i-1}}$ at time t_{i-1}. There is such a constraint for every voxel on and inside the guiding object at t_{i-1}. Note that the smoke transportation scheme in this section only accounts for large rigid-body motion and leaves nonrigid deformation to the velocity constraints in the previous section.

The velocity constraints we have come up need to be applied either at the boundary or interior of the smoke when the shape difference between the guiding object and the smoke region becomes sufficiently large. To make this statement more concrete, we are going to discuss in the following sections how to measure the shape difference and how to detect and evolve the smoke boundary.

3.3.3 Error Metrics

We use two different error metrics for measuring the shape difference. The velocity constraints are only applied when the specific metric being used exceeds some threshold. The first one is the volume discrepancy e_v defined in Eq. (3.1). The second metric is a generalization of the L^p norm to object boundaries,

$$e_{L^p} = (\int_{\Gamma_\rho} D^p(\mathbf{x}, t) d\mathbf{x})^{1/p} \tag{3.6}$$

where p is positive, Γ_ρ is the boundary of the smoke and $D(\mathbf{x}, t)$ is the distance function for the guiding object at time t. The latter two quantities have been defined in Section 3.1. A special case of the L^p norm

is the L^∞ norm, which is equivalent to

$$e_{L^\infty} = \max_{\mathbf{x}\in\Gamma_\rho} D(\mathbf{x}, t) \tag{3.7}$$

Although the velocity constraints were derived using the first metric, both metrics have the same global minimum and can effectively reduce the shape difference. However, the dynamic behavior of the smoke under their respective control can be quite different, especially when p becomes large in the second metric. For instance, e_v allows small portions of the smoke boundary to be far away from the guiding object as long as the integral remains small while e_{L^∞} keeps all the boundary points of the smoke within a certain distance from the guiding object. As a result, e_v or e_{L^p} with low p values give rise to more realistic smoke appearances on the boundary and less clear object structures while e_{L^p} with high p values becomes more appropriate when clear object structures are desirable.

3.3.4 Smoke Evolution and Boundary Detection

The smoke implicit function $\phi = \rho - \tau$ is evolved passively over time by the wind velocity field $\{\mathbf{u}\}$ which may be partially constrained. It can be easily shown [36] that the equation to update ϕ is as follows.

$$\phi_t + \mathbf{u} \cdot \nabla\phi = 0. \tag{3.8}$$

Except for the density threshold τ for boundary detection, this equation coincides with the advection equation for the smoke density in [29]. This equation can be solved either by the semi-Lagrangian method or by the upwind scheme [37]. Note that both operate on the whole voxel grid instead of boundary voxels only. We have implemented both methods. Both of them can produce visually realistic results. Since a smoke boundary tends to evolve relatively slowly, the choice of a numerical method is not very critical.

Our normal and tangential velocity constraints are positioned at the boundary of the smoke region. Enforcing these constraints requires the detection of the smoke boundary at every time step. This can be easily achieved since this smoke boundary is just the zero isosurface of the smoke implicit function.

3.4 Smoke Simulation for Constrained Velocities

Since we exploit velocity constraints to achieve shape matching, a smoke simulation framework that can effectively incorporate hard-wired velocity constraints without producing visual discontinuities is desirable.

3.4.1 An Empirical Equation for Compressible Gases

The basic framework in this work is a dynamic feedback system where the velocity constraints defined in Section 3.3 are dynamically updated every time step according to the shape discrepancy between the smoke region and the underlying guiding object. The purpose of such velocity constraints is to reduce the amount of discrepancy. These velocity constraints have poor spatial and temporal coherence because constraints at spatially or temporally adjacent voxels may be quite different. In addition, the constraints are only imposed at the boundary of the smoke region which is essetially a thin layer in the simulation volume. Since these artifical velocity constraints do not exist in the real world, we argue that existing fluid simulation methods cannot incorporate them without introducing artifacts.

First, the incompressible scheme in Eq. (2.5) and (2.6) does not work well. If we impose velocity constraints which generate discontinuities in the velocity field, concentrated high pressure tends to appear rapidly in regions nearby to guarantee zero divergence. Those sudden high pressures further influence surrounding velocities and generate temporal discontinuities and visual artifacts. Such artifacts have been observed in our experiments. Second, existing compressible schemes would not fit our purpose very well, either. Simulating true compressibility is very expensive and requires a strict time step condition. On the other hand, an unphysical scheme called "artificial compressibility" was introduced in [38]. It allows a certain degree of compressibility during a transition period, and converges to incompressible fluid simulation when a steady solution has been reached. However, this convergence is achieved over time instead of within each time step, and is only guaranteed when external forces are absent. In our simulation, incoherent external forces are exerted constantly at every time step since we need to update the velocity constraints. Thus, the scheme in [38] may not converge. In addition, the sound speed used in this scheme needs to be unusually small to prevent temporal discontinuities. An unreasonable sound speed further compromises the realism in the simulation results.

The nature of our control scheme demands a new fluid simulation technique which does not have to be physically accurate, but needs to produce results that are visually appealing. What we need is a simulation technique that approximates the behavior of incompressible fluids while absorbing discontinuities where velocity constraints are present. We would also like the technique to be stable and efficient. To achieve our goals, we propose an empirical scheme that does not strictly enforce incompressibility. Recall that the pressure P in (2.6) can be estimated numerically using the Poisson equation [28, 29]

$$\nabla^2 P = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}, \tag{3.9}$$

where Δt is the size of the simulation time step. It has been shown in [28] that imposing a feedback force field $-\nabla P$ to the velocity field strictly enforces zero divergence. Our technique first decomposes the pressure field into two components, $-\mu\Delta t\nabla^2 P$ and $P' = P + \mu\Delta t\nabla^2 P$, where μ is a constant factor with units of area divided by time, and the negative sign in front of the first component is due to the fact that a local maximum of P is typically a local minimum of $\nabla^2 P$. Note that the first component is the negative Laplacian of the pressure field, and the second one is a blurred version of the original pressure. This decomposition of the pressure field in turn splits the feedback force field into two components, $\mu\nabla(\Delta t\nabla^2 P)$ and $-\nabla P - \mu\nabla(\Delta t\nabla^2 P)$. To allow a certain degree of compressibility, especially in the regions with velocity constraints, our scheme only applies the first force component to the velocity field immediately while buffering the second one for later time steps. Thus, the discretized version of (2.6) changes to

$$\frac{\mathbf{u}(t+\Delta t) - \mathbf{u}(t)}{\Delta t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \mu\nabla(\Delta t\nabla^2 P) + \mathbf{f}. \tag{3.10}$$

Because of (3.9), $\Delta t\nabla^2 P$ in (3.10) can be replaced with $\nabla\cdot\mathbf{u}$. If we further replace the left hand side of (3.10) with a continuous partial derivative of the velocity field, we arrive at the following single partial differential equation that is capable of simulating compressible gaseous phenomena:

$$\frac{\partial\mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \mu\nabla(\nabla\cdot\mathbf{u}) + \mathbf{f} \tag{3.11}$$

where $0 \le \mu < \infty$ with a typical value between 0 and 1. It is used to adjust the magnitude of the feedback force from the pressure field to the velocity field.

Since our scheme only uses a filtered version of the original pressure every time step, the feedback force from the pressure to the velocity field is weakened. Nevertheless, this does not mean a portion of the pressure simply disappears, but mean that the release of the energy preserved in the rest of the pressure is delayed. In Appendix B, we show that, for a bounded workspace without sources and sinks, $\nabla(\nabla\cdot\mathbf{u}) = 0$ everywhere is equivalent to $\nabla\cdot\mathbf{u} = 0$ everywhere. This result indicates that our new formulation of the feedback force from the pressure to the velocity field can eventually reduce the divergence of the velocity to zero when there are no external forces. In addition, our formulation allows large time steps and does not involve a sound speed. While other decompositions are possible, the Laplacian operator in our pressure decomposition makes it particularly convenient to robustly obtain the components through a diffusion process, which will be discussed in Section 3.5.2.

In practice, we have found that our formulation can effectively reproduce realistic fluid motion as well as incorporate frequently inserted velocity constraints without generating obvious visual discontinuities (see

Section 3.6.1).

3.5 Implementation

3.5.1 Shape Transition

Variational Shape Morphing and Interpolation

As our first option, we apply the shape transformation method introduced in [39] to generate a morph sequence between the shape of a smoke region and a target object. This method represents the whole morph sequence as a variational implicit function defined in a space of $n + 1$ dimensions. This implicit function, which is based on radial basis functions, interpolates the source and target shapes defined in an n-dimensional space and handles topological shape changes automatically. The extra dimension is aligned with the temporal axis so that the source and target shapes are n-dimensional slices of the implicit function at time zero and one, respectively. In our situation, we apply this method to obtain an interpolating 4-D function $\Phi(\mathbf{r}, s)$ such that $\Phi(\mathbf{r}, 0)$ and $\Phi(\mathbf{r}, 1)$ reproduce the starting smoke shape and the target object, respectively. By fixing s to a value between 0 and 1, $\Phi(\mathbf{r}, s)$ represents the analytic from of an intermediate shape. In practice, we found that the rate of shape transformation usually was not uniform and the target shape started to loom only when the time becomes very close to one if the target shape is complicated and the parameter s is scheduled linearly with respect to the actual time scale. Therefore, we decided to warp the temporal axis and schedule s as a piecewise linear function of the actual time t used for an animation. The local slope of the piecewise linear function is used to adjust the transformation rate and make it perceptually more uniform. Better morph sequences can be obtained when the source and target objects are well aligned in terms of position and orientation. The amount of relative translation and rotation should then be uniformly distributed back across the frames of the resulting sequence so that there is simultaneous deformation, translation and rotation from frame to frame. It is desirable for the intermediate shapes to have an approximately constant volume since a gas usually does not significantly change volume even when it is compressible. Therefore, we verify the volume at each intermediate frame, and apply morphological operations on the voxel grid to shrink or expand the shape.

The same variational implicit functions for shape transformation can be used for surface interpolation [40, 41]. Therefore, they can be used for generating an analytic implicit function to approximate a polygonal mesh model by interpolating the vertices of the mesh. The interpolating implicit function and the polygonal mesh have the same dimensionality. Guiding objects originally represented as polygonal meshes are converted into implicit functions in this way.

Shape Masking

Shape masking is an easier-to-implement alternative for generating shape transitions. In this scheme, the target object can be simply a fixed shape undergoing rigid-body motion. We also require an additional masking shape, $M(\mathbf{x}, t)$, whose scale, position and orientation can all vary with time. The guiding shape at each time step is defined to be the intersection between the target and masking objects. For example, the masking object can be a moving sphere with changing radius. At the beginning of a transition, the sphere has a tiny radius and does not have any overlap with the target object. The sphere then moves closer to the target object while increasing its radius so that the amount of overlap becomes larger. At the end, the sphere encompasses the target object and the transition is complete. One caveat with this scheme is that the volume of the guiding shape changes with time, which requires an increasing amount of smoke during the transition.

3.5.2 Numerical Smoke Simulation

We have implemented numerical solutions for two formulations, our new compressible fluid formulation and the incompressible and inviscid formulation in [29]. We also implemented enhancements that allow the integration of our velocity constraints. The details of these enhancements will be introduced in the next section.

To simulate our compressible fluid formulation, we still keep the first step unchanged, but revised the other two. Although our formulation does not involve pressure, directly solving Eq. (3.11) may not be stable. Introducing pressure during the numerical process improves stability. Therefore, we actually solve Eqs. (3.9) and (3.10) instead. In the above steps, after obtaining the pressure by solving the Poisson equation, we apply a diffusion process,

$$\frac{\partial P}{\partial t} = \mu \nabla^2 P, \tag{3.12}$$

and solve for the new pressure, P', using a stable implicit method,

$$\frac{P' - P}{\Delta t} = \mu \nabla^2 P' \tag{3.13}$$

where the Laplacian of the new pressure $\nabla^2 P'$ instead of $\nabla^2 P$ appears on the right hand side. The new pressure can be obtained by solving the sparse linear system arising from discretizing this equation. Finally, the intermediate velocity is updated as follows,

$$\mathbf{u} = \mathbf{u}^* + \mu \Delta t \nabla (\nabla^2 P'). \tag{3.14}$$

Note that solving the diffusion equation to obtain the new pressure is an extra step we performed during smoke simulation.

Our overall numerical solution involves multiple (around three) time steps between two consecutive frames. Small and large object motion tracking are carried out in separate time steps. There is at most one optional time step dealing with large motion between two frames. And there may be one or more time steps for small motion tracking since multiple iterations may be necessary to achieve good boundary matching. In the following, we introduce the details of these time steps.

A time step for small motion tracking has the following substeps:

- Compute the intermediate fluid velocity field $\{u^*\}$, detect the boundary of the smoke region using a density threshold and set up velocity constraints at the boundary by modifying $\{u^*\}$.
- Solve the final fluid velocity field using (2.9) and (2.10) while satisfying the constraints.
- Use the obtained velocity field to evolve the smoke density distribution applying the semi-Lagrangian method.

The optional time step for large motion between two frames is executed before the other time steps. It has the following substeps:

- Obtain the intermediate fluid velocity field and set up velocity constraints for large motion.
- Solve the final fluid velocity field while satisfying the constraints.
- Use the obtained velocity field to evolve the smoke outside the guiding object; use the guiding object to transport the smoke inside and the transported smoke density overwrites existing density at each destination voxel.

If we choose to apply our compressible fluid formulation, the second substeps in both types of time steps should solve equations (2.9), (3.13) and (3.14) instead. It is straightforward to apply additional vorticity confinement [29] in the second substeps as well since these substeps are basically the major part of a single forward simulation step. In our experiments, we always apply vorticity confinement.

3.5.3 Enforcing Velocity Constraints

Most of the time, we only set up and enforce velocity constraints at a selected subset of the voxels on the smoke boundary for small motion tracking. This process is initiated once the adopted error metric (see Section 3.3.3) exceeds a prescribed threshold. A larger threshold produces more lively smoke motion while

a smaller threshold shows object boundary more clearly. The voxels on the smoke boundary are sorted according to their distance to the guiding object. Voxels with larger distance are assigned with higher priority. Velocity constraints are assigned to the boundary voxels with highest priority first. If the shape discrepancy does not drop significantly, additional voxels with next level of priority become constrained. This process continues until the error drops below the threshold or most of the boundary voxels are constrained. Once the error becomes sufficiently small, constrained voxels get released gradually in the reversed order.

The desired velocity constraints are imposed after the computation of the intermediate velocity field $\{\mathbf{u}^*\}$. Solving the Poisson equation may alter the velocities in these constraints and make it difficult for the smoke to converge to the target shape if these constraints are not explicitly enforced. Enforcing these constraints means the constrained velocities should not be affected during this step. Eq. (2.10) indicates that ∇P should be zero at the constrained voxels. Therefore, the pressure p needs to satisfy such gradient constraints in addition to the Poisson equation.

Using finite differences to discretize both left and right hand sides of (2.9), we obtain a sparse linear system of equations, $\mathbf{AP} = \mathbf{b}$ where $\mathbf{P}$ is the vector of unknown pressures and $\mathbf{A}$ is the coefficient matrix with a sparse structure. A discrete gradient estimation at a voxel involves three or six of its direct neighbors depending on whether central differences are being used. For symmetry consideration, when $\nabla P = 0$ at voxel (i, j, k), we impose that $P(i, j, k)$ should equal the pressure at its six direct neighbors. For every equation involving $P(i, j, k)$ in the linear system, we need to replace $P(i, j, k)$ with the pressure at one of its direct neighbors. Eventually, we can eliminate $P(i, j, k)$ from all the equations. Changes like this result in a linear system with a reduced number of unknown variables.

Since the smoke boundary surface is typically a closed surface, we avoid strictly enforcing boundary conditions everywhere on the surface because that would result in a singular matrix $\mathbf{A}$ if the volume inside the smoke region is not preserved. When it is necessary to cover the whole smoke boundary with constraints, we reduce the density of the constraints by setting a small distance threshold d_t. Any voxels within that distance from a constrained voxel are not allowed to be constrained. We typically choose d_t to be between 1 and 3 voxels. Alternatively, Dirichlet boundary conditions can be adopted by directly specifying pressure values on the smoke boundary. For example, one can force voxel (i, j, k) and all its direct neighbors to have zero pressure.

3.5.4 Smoke Transportation for Large Motion

We assume that the frame-to-frame motion of the guiding object can be decomposed into a rigid-body transformation and a nonrigid deformation. Smoke transportation only handles large rigid transformation

and leaves small rigid transformation as well as additional nonrigid deformation to small motion tracking. To move the smoke to a new position and orientation specified by the rigid transformation, we apply backward mapping with a modified trilinear interpolation scheme. A voxel inside the transformed object takes the smoke density at a corresponding position in the original object. The density at the corresponding position should be interpolated only from the densities at those surrounding voxels that are also inside the original object to preserve the object boundary.

3.5.5 Thin Parts

It is hard for the smoke surface to reach some of the thin parts of the guiding object. That is because the gradient and curvature estimations are inaccurate at these locations due to insufficient sampling. We explicitly label those voxels that should have smoke but actually does not after a long time. If there is a smoke-filled voxel which is adjacent to one of those tagged voxels, we set a velocity constraint at the first voxel and the velocity points to the tagged voxel. Thus, smoke can be propagated gradually into the thin parts.

3.5.6 Time Complexity

Since we enforce shape matching by embedding velocity constraints into a conventional smoke simulation, the time complexity of our approach is slightly higher than the latter. The extra work at each time step includes obtaining the signed distance function of the guiding object and solving the diffusion equation for the pressure among others. The complexity of the distance transform is $O(n \log m)$ where n is the number of voxels in the grid and m is the maximum number of voxels on the propagating front [37]. In the worst case, $m = O(n)$ which usually does not occur unless the object boundary becomes close to a fractal surface. In all our experiments, the actual running time of the distance transform is less than an original smoke simulation step without constraints. The complexity of solving the sparse linear system arising from the diffusion equation is on the same order as that of solving the Poisson equation which is part of the original simulation. Solving these two equations is the most time-consuming step which occupies approximately 60% of the total running time. The complexity of all the other extra work is linearly proportional to the size of the voxel grid. Therefore, in practice, our complete algorithm maintains the same order of magnitude of the original complexity of smoke simulation.

3.6 Results

We have successfully tested our complete algorithm in a variety of examples. Some of the images from these examples are shown in Fig. 3.1 and Fig. 3.4-3.8. The target objects in these examples include both free-form objects and letters. Most of the target objects are converted to implicit functions from triangular meshes. If there are multiple objects in the same example, smoke simulation was performed on all of them simultaneously. Most of the simulations have been finished on an AMD 2100+ processor. A voxel grid of size between 64^3 and 200^3 has been adopted for the simulations. The examples took 15 seconds/frame on the lower end of the grid resolution and 60 seconds/frame on the higher end. The final images were rendered by ray-tracing the smoke volume density distribution [42].

The density threshold τ for smoke boundary detection is interactively determined at the beginning of the smoke-object shape transition, and fixed throughout a whole simulation. We have found that typically this threshold falls between 0.1 and 0.5. In the following, we specify other parameters used during experiments for a normalized grid with one unit length for each of its dimensions. In (3.4), we typically set d_{max} between 0.05 and 0.1, and set C_n around 25. In (3.5), we set C_t around 2. These parameters were chosen with the assumption that the maximum velocity of the smoke is around 1 unit length per second. Obviously, by adjusting the maximum velocity, we can change the overall pace of the smoke.

3.6.1 Validation of the Compressible Fluid Model

We have validated our empirical fluid model by visually comparing the results from our model with results from the model in [29]. We have performed two different types of comparisons. The first set of comparisons are between smoke simulations without velocity constraints. They are meant to verify whether our model can be used for a general smoke simulation as opposed to the special application in this work. As a result, the visual quality of the smoke sequences generated by our model is comparable to those in [29]. It can produce realistic smoke appearances with rolling effects. Fig. 3.3 shows one comparison in this category.

The second type of comparisons are between simulations with our derived velocity constraints. We have confirmed from these comparisons that our new model can integrate these velocity constraints better with much less visual artifacts. Introducing our velocity constraints into a model enforcing incompressibility would generate obvious artifacts since the nonzero divergence caused by the constraints influence surrounding velocities immediately. The reason our model can perform better is that it can absorb velocity discontinuities into pressure and release the energy in the pressure gradually. A comparison can be found in the accompanying video since these artifacts are easily visible in animations, but not in still images.

3.6.2 Interaction with the Environment

The smoke objects should interact with the environment like smoke. When there is a sufficiently strong wind, the animator can choose to release the velocity constraints in the influence region of the wind and let the smoke move freely. When there is an intervening real object, the animator can release the original velocity constraints in the contact region and let the smoke be controlled by the boundary conditions on the real object surface as in [29]. In the example shown in Fig. 3.1(a), a strong wind blows away the head of a smoke horse. However, the head grows back when the wind recedes. This is because we keep the underlying target object unchanged, and the target object induces velocity constraints pointing towards the head region. In such examples, the animator needs to specify the spatial region where constraints should be released as well as the starting time and duration. In Fig. 3.1(a), the user simply places a partitioning plane at the bottom of the horse's neck. Constraints on the same side of the plane as the head are released for a short period of time. To preserve temporal coherence, the number of constraints can be decreased or increased gradually to produce a smooth transition.

3.6.3 Interaction between Smoke Objects

When two smoke objects collide, their density distributions overlap and they should naturally merge into a larger smoke region since they do not have hard boundaries. Other scalar fields, such as temperature and color, can also be advected by semi-Lagrangian tracing and blended together in the same way as the density. The underlying guiding objects should either be merged or be replaced with a new one which may require a new shape transition. Velocity constraints should only be imposed on the new boundary afterwards. In the example shown in Fig. 3.4, two heart-shaped smoke objects merge into a larger one and their original colors get advected and blended together. The heart-shaped target object is defined by a closed-form implicit function: $(2x^2 + y^2 - z^2 - 1)^3 - 0.1x^2z^3 - y^2z^3 = 0$.

3.6.4 Large Motion

To demonstrate large motion tracking, we use a synthetic character equipped with motion-captured data as the underlying guiding object. The limbs of this articulated character have large frame-to-frame motion so that a small amount of smoke escapes. We also put a few smoke sources on the floor. The results from this example are shown in Fig. 3.5. In general, smoke does not move very fast. Its appearance would look less natural if we force the smoke to follow fast motion. Nevertheless, it can be clearly seen that the velocity constraints induced by the fast motion of the limbs creates an interesting velocity field for the rest of the environment and the smoke around the character follows this velocity field.

3.6.5 Tradeoff between Control and Smoke Appearance

The number of velocity constraints on the smoke boundary is the most important factor affecting the realism of the smoke simulation because our velocity constraints are kinematic constraints that do not involve dynamics. The shape of the smoke region is free to evolve without any constraints, but is guaranteed to match the guiding object with dense constraints. Thus, we can achieve various levels of tradeoff between control and smoke appearance by changing the number of constraints. However, the user does not have direct control over the number of constraints which is automatically adjusted at each time step according to the error metrics, the error thresholds and other parameters, as discussed in Section 3.3.3 and 3.5.3. There are no velocity constraints at all if the shape discrepancy is below the error threshold(s); otherwise, the number of constraints is increasing and realism is partially given up for control. Therefore, in general, larger error thresholds allow more realistic smoke appearance, but less object structures. The parameter μ in Eq. (3.11) can also affect the liveliness of the smoke. Larger μ's provide stronger force feedback from the pressure and typically lead to more lively motion. Fig. 3.9 gives a comparison of the visual quality generated by different combinations of error metrics, thresholds and μ. In this experiment, we chose to use both e_v and e_{L^∞} simultaneously. For e_v, the threshold is set between 0% and 5% of the volume of the target object. For e_{L^∞}, the error threshold is set between 0 and 4 voxels in a 128^3 grid. μ is set between 0.1 and 0.3.

3.6.6 Comparisons with other Control Schemes

A simple solution to our problem is to perform smoke simulation in the volume enclosed by the boundary surface of the guiding implicit function. Obviously, the smoke is forced to follow the object when it moves. This solution is similar to our scheme for large motion, but the hard boundary used for large motion disappears once the smoke has been moved to the target location. There are a few limitations with this solution using a hard boundary. Even though the boundary surface is not visualized during final rendering, it will still be obvious since the smoke stops or reflects at the boundary. A smooth hard boundary for the smoke is not a realistic phenomenon. The second problem is that the smoke tends to be stationary without interesting motion inside a closed volume with a limited size. A side-by-side comparison is made between this simple scheme and our method in Fig. 3.6. In the results generated by our method, the smoke close to the boundary of the target object still has natural motion and advection.

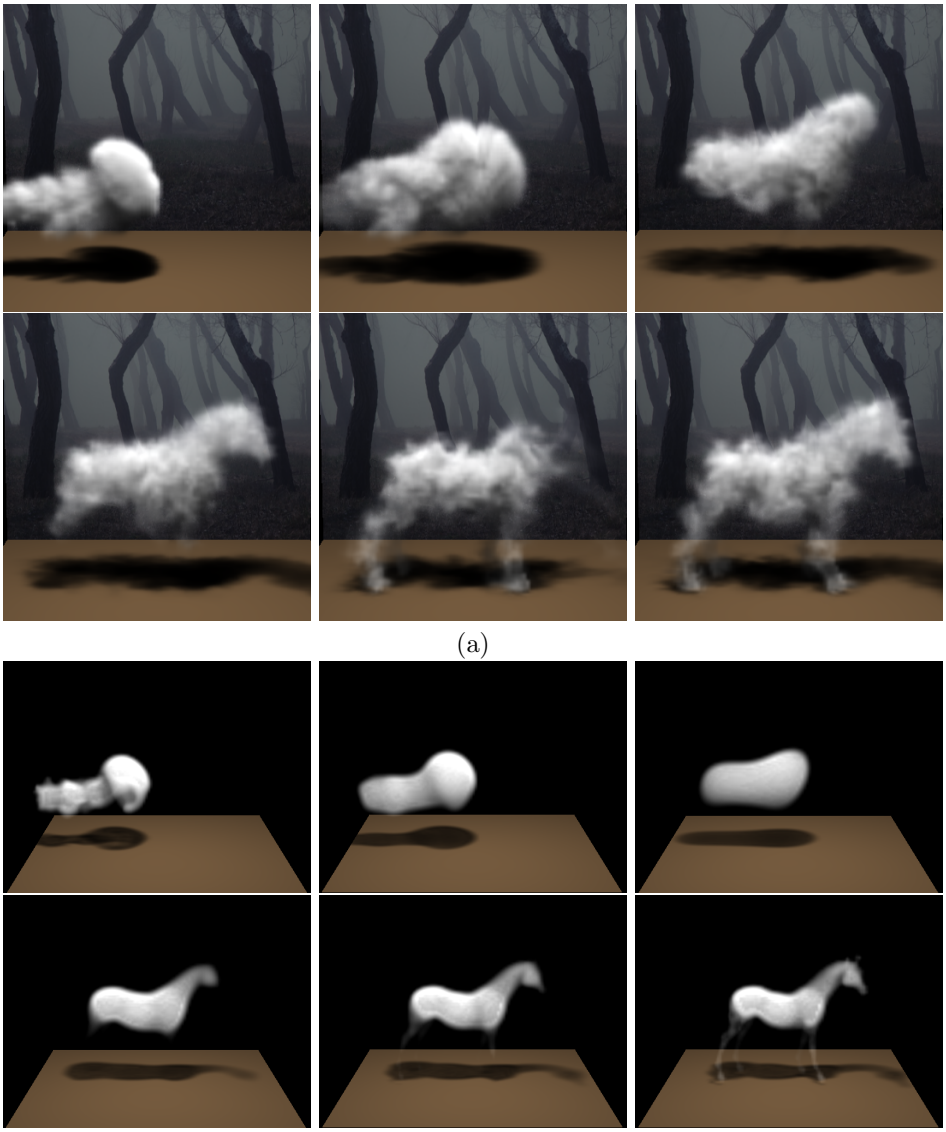
Our control scheme also has advantages and differences if compared to the method in [15, 16]. First, our scheme is more efficient by introducing velocity constraints instead of evaluating derivatives of the velocity field with respect to all control parameters. The time complexity of our method is on the same order of magnitude as a single smoke simulation without shape matching. For example, Fig. 3.7 shows an example

where the smoke switches back and forth twice between the shapes of a "check" and a "X" on a 128^3 grid. It took only 4 hours 15 minutes to generate the 1250-frame sequence. Second, our scheme can maintain dynamic smoke appearances around a shape for an arbitrarily long time. Nevertheless, if we release the velocity constraints, the smoke becomes free to evolve (Fig. 3.8(d)). As shown in Fig. 3.1(a), we can also release some of the constraints and then regain control as we wish. A notable difference is that our algorithm is targeted at object shapes while the method in [15] is designed for arbitrary density distributions at keyframes.

3.7 Conclusions and Future Work

We have presented a novel technique to control the density and dynamics of smoke (a gas phenomenon) so that the synthetic appearance of the smoke (gas) resembles a still or moving object. The main focus has been on controllability and appearance. In order to match the smoke surface with the target surface, we represent both smoke and objects as implicit functions and impose carefully designed velocity constraints derived from a shape matching functional. An empirical compressible fluid model has been introduced for effectively integrating constraints into the velocity field while maintaining realistic fluid appearances. The overall framework represents a significant advance over previous methods for controlling fluids. The implementation of our system is built upon recent advances in smoke simulation and shape transformation.

We would like to extend this work to 2D images and video objects. The difference between a 3D object and a 2D image is that an object is a binary function while an image has multiple levels of intensities or colors. Matching the smoke boundary with the object boundary should be extended to matching the level sets of smoke density with the level sets of image intensities or colors. Nevertheless, we expect the approach introduced here can be generalized to handle that circumstance by matching the boundaries of these level sets. Unlike 3D animated objects addressed in this work, the motion of video objects is unknown, and needs to be solved using computer vision techniques such as optical flow [43, 44]. Once the motion of these objects has been estimated, the approach in this work can also be adapted to produce fluid appearances for them.



(a)

(b)

Figure 3.1: (a) An initial smoke blob evolves into a smoke horse. A wind blows the head away. When the wind recedes, the head grows back. (b) The underlying shape transformation for the smoke-horse transition.

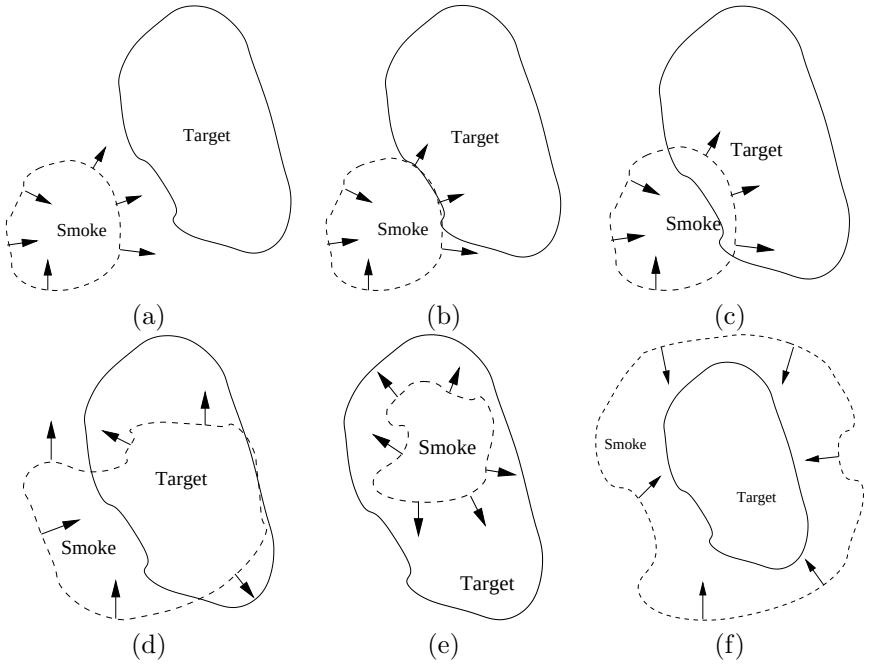


Figure 3.2: Normal velocity constraints on the smoke boundary for various situations. They are based on our revised minimization scheme for Eq. (3.2). (a) The smoke region is completely outside the guiding object; (c) the smoke region partially overlaps with the guiding object; (b)&(d) a portion of the smoke boundary touches the boundary of the guiding object. The smoke velocity should not be affected in (b) while it should be reduced to zero in (d); (e) the smoke region is completely inside the guiding object; (f) the smoke region encloses the guiding object.

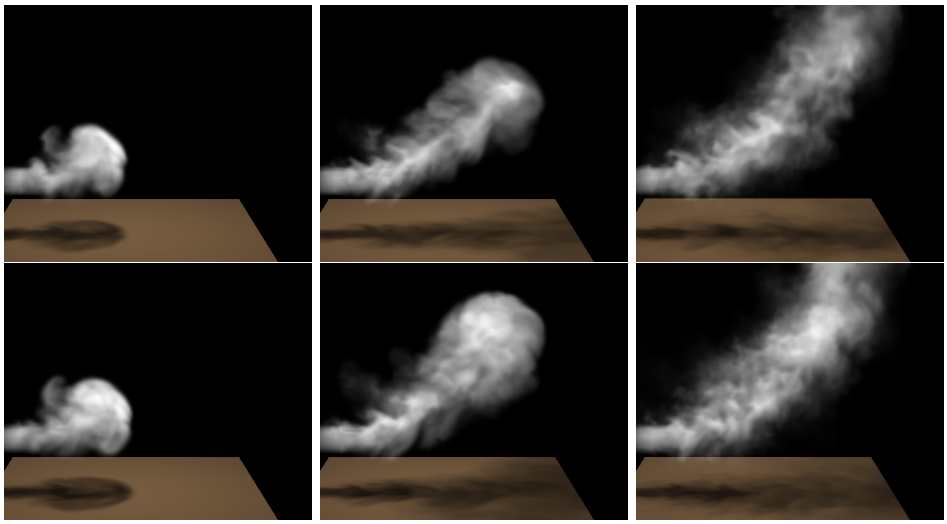


Figure 3.3: A comparison between two smoke formulations. The first row is generated by a model enforcing incompressibility. The second row is generated by our model with compressibility. They are visually similar, which indicates our model can also produce realistic smoke appearances. Grid size 200^3.

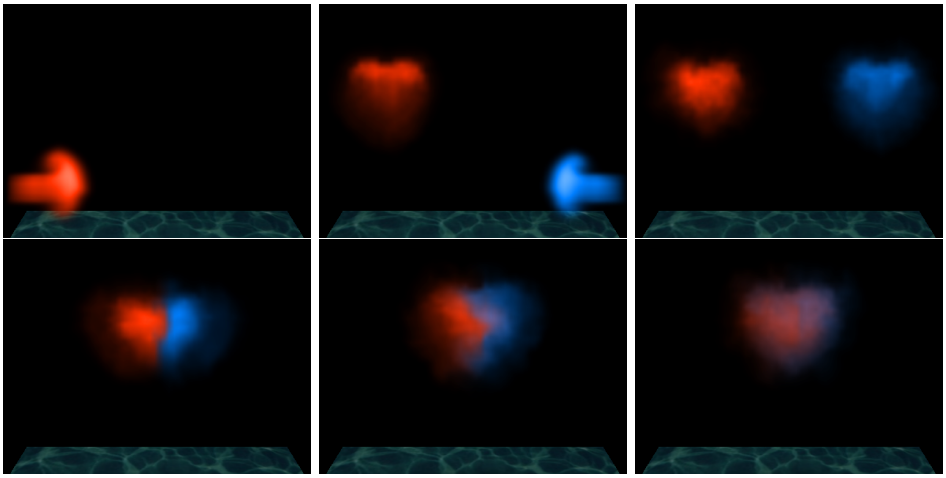


Figure 3.4: Two heart-shaped smoke objects collide and merge into one. Grid size 128^3.

Figure 3.5: Large motion tracking for a synthetic character. Constraints for large motion induce a velocity field for the environment. There are a few sources on the floor to produce smoke for the environment. Grid size 128^3.

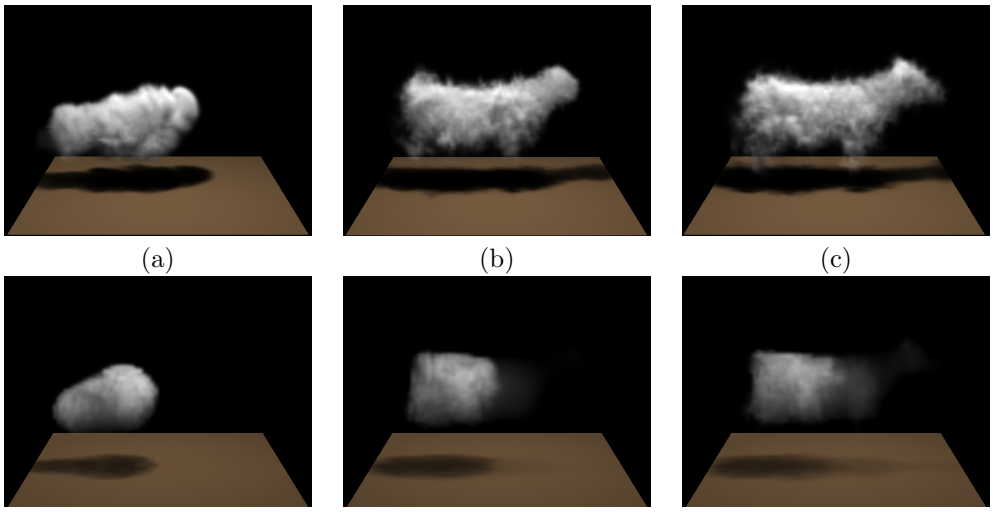


Figure 3.6: (a)-(c) A smoke cow generated by our method; (d)-(f) a smoke cow generated by the hard boundary scheme. Grid size 128^3.

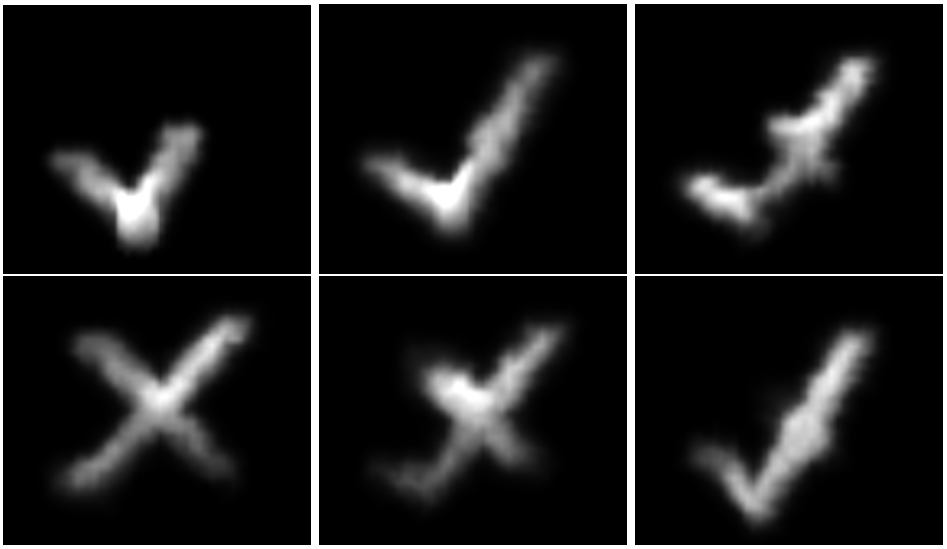
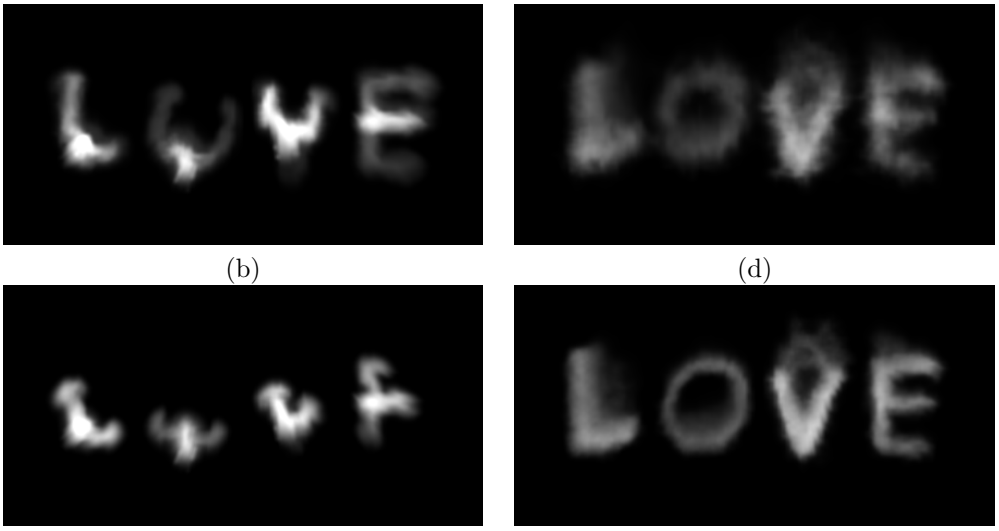


Figure 3.7: The smoke switches back and forth between the shapes of a "check" and a "X". Only one cycle is shown here. Grid size 128^3.



(b) (d)

(a) (c)

Figure 3.8: (a)-(d) Smoke rises to form the shape of four letters. The shapes start to disappear once the velocity constraints are released in (d). Grid size 128^3.

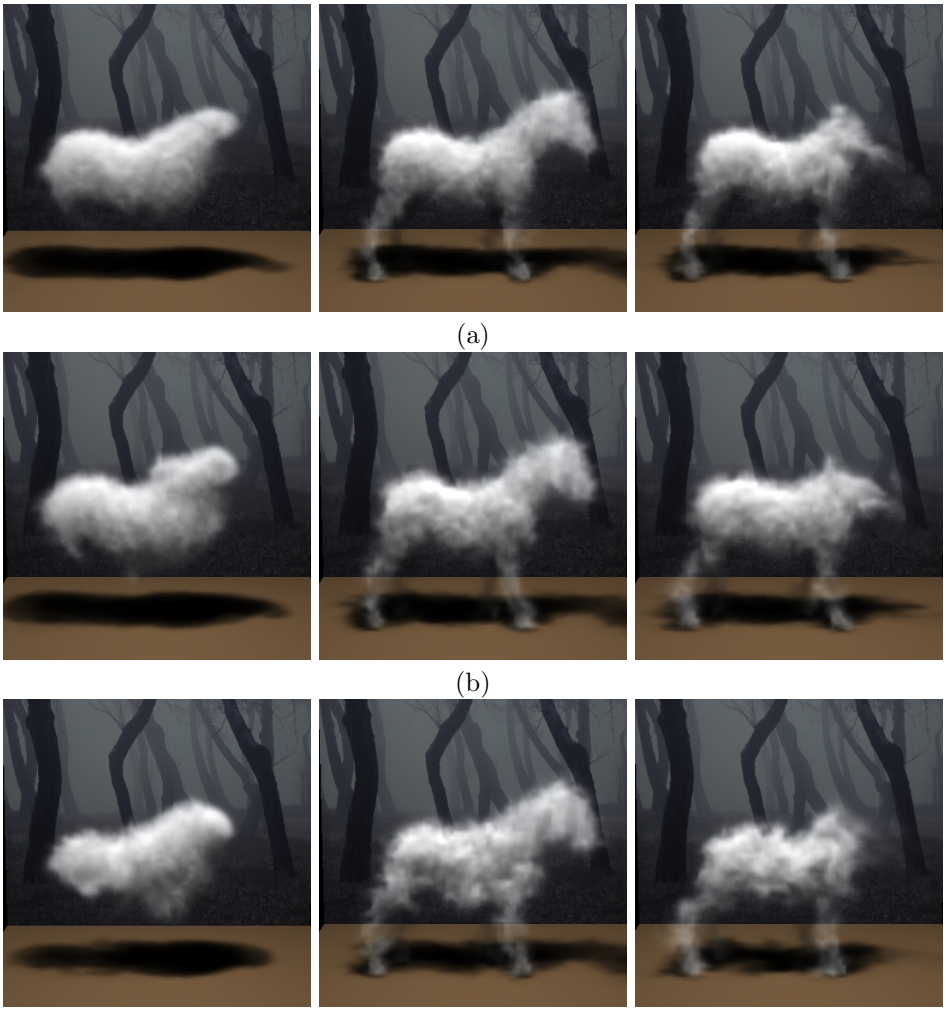


Figure 3.9: A comparison of the visual quality of the smoke objects by different combinations of error metrics, thresholds and the pressure diffusion parameter. (a) The error thresholds $e_v = 0\%$, $e_{L^\infty} = 0$ voxels, and the diffusion parameter $\mu = 0.1$; (b) $e_v = 3\%$, $e_{L^\infty} = 2.5$ voxels, and $\mu = 0.1$; (c) $e_v = 5\%$, $e_{L^\infty} = 4$ voxels, and $\mu = 0.3$. Grid size 128^3.

Chapter 4

Taming Liquids for Rapidly Changing Targets

Following rapidly changing target objects is a challenging problem in fluid control, especially when the natural fluid motion should be preserved. The fluid should be responsive to the changing configuration of the target and, at the same time, its motion should not be overconstrained. In this chapter, we introduce an efficient and effective solution by applying two different external force fields. The first one is a feedback force field which compensates for discrepancies in both shape and velocity. Its shape component is designed to be divergence free so that it can survive the velocity projection step. The second one is the gradient field of a potential function defined by the shape and skeletion of the target object. Our experiments indicate a mixture of these two force fields can achieve desirable and pleasing effects.

4.1 Introduction

Recent progress on physics-based fluid simulation in the graphics community has produced stunning dynamics comparable to their counterparts in the real world. Nevertheless, animators not only need realism, but also need to achieve certain design goals when producing animations. In the past, liquid actors and animals, such as those in "The Abyss" and "Terminator 2: Judgment Day," have been created with realistic rendering, but without the desirable liquid dynamics. Recent techniques on liquid control [16, 45] have incorporated physics-based fluid dynamics. But they are not particularly convenient to use. Fluid control methods should be evaluated with the following criteria:

- **Control capability.** When given a set of user-specified constraints, such as key frames or target shapes, a control method should be able to force the movement of the fluid to approximately satisfy the constraints.
- **Ease to use.** As an authoring tool, the method should be able to produce desirable fluid animations without too much computational cost or user intervention.

Figure 4.1: A controlled simulation with a water horse emerging, running and collapsing.

- **Fluid-like motion.** The control method should not overconstrain the fluid. The natural movement of the fluid should be preserved as much as possible.
- **Stability.** The controlled movement of the fluid should be stable without obvious oscillations.

In this chapter, we focus on controlling liquids to match rapidly changing target shapes which represent regular non-fluid objects. In other words, we assume the animator prepares a continuous sequence of frames instead of sparse keyframes. This is a somewhat different goal from previous work, and is particularly useful when the whole sequence of target shapes is essential to represent the target motion such as a running water man. The physical properties of liquids and the spatially varying large velocities and accelerations over the target shape pose challenges on the control method. First, the large magnitude of the velocities and accelerations of the target shape demands a stable and well-balanced fluid control method to track the fast-changing global shape while maintaining natural fluid motion. Second, when compared to gases, liquids have a clear boundary at their interface with the air and their volume is strictly incompressible. These

properties imply that undesirable artifacts introduced by a control method can be easily noticed.

With the unique challenges posed by the aforementioned problem, we present a simple but effective control technique by applying two different external force fields.

- The first one is a feedback force field which compensates for discrepancies in both shape and velocity (Section 4.3). Its shape component is designed to be divergence free so that it can survive the velocity projection step. Its velocity component can effectively prevent overshooting and improve stability.
- The second one is the gradient field of a potential function defined by the shape and skeletion of the target object (Section 4.4). We choose to use a nonlinear function of the signed distance transform of the target shape as the potential so that large deviations will be reacted to more strongly.

Our fluid control method was inspired by the optimal control theory in [46] which is based on both potential shaping and feedback control.

Our control method meets all of the aforementioned criteria. It is capable of tracking rapidly changing targets while maintaining fluid-like motion. Given a target animation, computing the spatially varying feedback and potential forces are automatic and fast. There are only three tunable parameters representing the overall strengths of the force fields. Thus, our control technique does not incur much extra computation and user intervention beyond a regular fluid simulation. As a result, it typically only takes a few hours to author a fluid animation with controlled behavior.

4.2 Related Work

This work has been made possible by much of the previous work on fluid simulation [28, 29, 11, 47, 48, 49, 50], and the level set method [37, 51]. In particular, our liquid simulation solves the Navier-Stokes equations and follows the particle level set implementation in [52]. In the following, we are going to focus on fluid control methods which are most relevant to this work.

Foster and collaborators initiated fluid control in graphics. Foster and Metaxas čiteFM97 proposed embedded controllers that allow animators to specify control parameters in a fluid animation. In [11, 12], animator-designed space curves and surfaces can be applied to control the motion and structures of fluids. The tangents of the curves or the normals of the surfaces indicate the directions of motion. In light of this approach, a control method based on relatively dense particles (one particle per grid cell) [53, 45] has produced impressive results. The control particles can not only control the velocity of the fluid in their local neighborhoods, but also control viscosity, the level set of the liquid surface, and the velocity divergence. Although the results of production quality, the keyframes for all the particles need to be generated from an

animation package, which involves a significant amount of work from an animator. These techniques do not allow the user to directly specify higher-level objectives, such as matching the density or shape of a target object.

Recently, a systematic optimization framework for controlling both smoke and liquid simulations through user-specified keyframes was introduced in [15, 16]. By viewing fluid simulation as a composition of functions, simulation derivatives can be surprisingly obtained from the fluid solver itself. However, since the derivatives with respect to each control parameter need to be computed throughout an entire simulation, this approach is very expensive. The adjoint method was adopted in [16] to significantly improve the efficiency of these derivative evaluations for each iteration during the nonlinear optimization. Nevertheless, a number of iterations are still necessary to obtain a good solution. As a result, this framework is not particularly animator friendly since the optimization takes a long time and can only be performed on relatively coarse grids. Furthermore, the objective function of the optimization forces the fluid to tightly match the target density, sacrificing its natural appearances. The method introduced in this chapter is much more efficient and can handle grids with two orders of magnitude more voxels. In addition, our method can maintain much of the natural motion of fluids.

An efficient and novel technique to match smoke density against user-specified distributions was reported in [14], which involves a custom-designed driving force term and a smoke gathering term. It has demonstrated much shorter computing time than nonlinear optimization. However, the driving force term requires a nonuniform density distribution for its gradient estimation. Meanwhile, another efficient method based on the level sets of the smoke was introduced in [6]. It can effectively control the smoke shape without the forcing and gathering terms in [14]. However, it relies on a compressible fluid model to eliminate visual artifacts caused by the constraints on the smoke boundary. Such a compressible model becomes inappropriate for liquids which are strictly incompressible. The method in this chapter applies a divergence-free feedback force field for shape differences to avoid this problem. It also applies additional velocity feedback forces to achieve stability.

Yet another simple and clever control scheme for smoke simulation was presented in [5], which exploits a potential function based on the shape of the target object. The negative gradient of the potential function serves as an extra force field which tries to reduce the overall potential of the smoke region. Upon its convergence, the shape of the smoke region coincides with the target shape, which represents the lowest energy configuration of the potential function. The original paper [5] focuses on smoke and only has results for static target shapes. When the target shape has complex and rapidly changing motion, it is unclear how to keep the liquid in pace with the target. In addition, when the configuration of the liquid is inconsistent

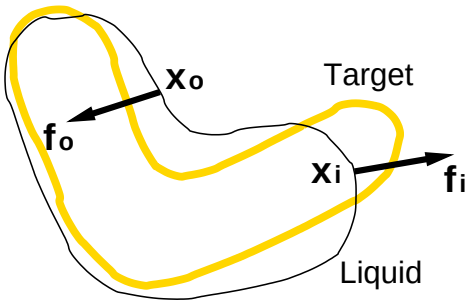


Figure 4.2: The initialization of the shape feedback force on the liquid interface. The initial force $\mathbf{f_o}$ at $\mathbf{x_o}$ outside the target object is defined according to (4.3); $\mathbf{f_i}$ at $\mathbf{x_i}$ inside the target is defined according to (4.4).

with the target shape, undesirable oscillations around the target shape become obvious because of the inertia of the liquid. The method in this chapter overcomes these problems.

4.3 Feedback Control Forces

During a controlled liquid animation, our algorithm applies a feedback control force field to reduce the shape discrepancy between the liquid and the target. Since the target surface may have large accelerations, control forces only based on shape differences become insufficient to guarantee both convergence and natural motion. Inspired by control theories, such as [54, 46], we decided to apply feedback forces to compensate for velocity differences as well. As a result, the complete feedback force has two terms,

$$\mathbf{f}_{feedback} = \mathbf{f}_{shape} + \mathbf{f}_{velocity}, \tag{4.1}$$

where $\mathbf{f}_{shape}$ compensates for shape differences and $\mathbf{f}_{velocity}$ compensates for velocity differences. In control terminology, $\mathbf{f}_{shape}$ corresponds to proportional control and $\mathbf{f}_{velocity}$ corresponds to derivative control. There is no obvious definition of a force field corresponding to integral control in the context of shape matching. Therefore, we replace the role of integral control with a geometric potential field which will be discussed in Section 4.4.

At any moment t, the liquid boundary and target shape are represented by two signed distance functions, $d_L(\mathbf{x}, t)$ and $d_T(\mathbf{x}, t)$, respectively. $d_T(\mathbf{x}, t)$ is positive when $\mathbf{x}$ is outside the target shape, and $d_L(\mathbf{x}, t)$ is positive when $\mathbf{x}$ is outside the liquid volume.

4.3.1 Velocity Feedback

To effectively apply this feedback force, we design a force field throughout the liquid volume. At any point $\mathbf{x}$ either inside the liquid volume or on the liquid boundary, the velocity feedback force is simply

$$\mathbf{f}_{velocity}(\mathbf{x}) = -\beta\,(\mathbf{v}_L - \mathbf{v}_T), \tag{4.2}$$

where β is the gain for derivative control, $\mathbf{v}_L$ represents the velocity of the liquid at $\mathbf{x}$ and $\mathbf{v}_T$ represents some target velocity at the same point. We use $\beta = 25$ in our experiments.

Ideally, the target velocity, $\mathbf{v}_T$, should be the velocity of the target shape. When a point on the liquid boundary is outside the target shape, the velocity of the target shape at that point is not well defined. Nevertheless, the signed distance function of the target shape is computed everywhere, and it moves with the target shape. We use the signed distance function to propagate the velocities of the target shape to all the points in the work space [55]. Thus, a point outside the target shape is assigned the velocity of the closest point on its surface. In our experiments, such a propagation produced desirable target velocities. For interior points of the target shape, usually this velocity can be uniquely determined. For example, the pointwise velocity of a rigid body can be obtained from its angular velocity and the velocity of its center of mass. In our experiments, articulated bodies are modeled as a collection of rigid segments. Given an interior point, we first determine which rigid segment it belongs to and then compute its velocity. For objects with a deformable surface, we assume the vertices of the surface correspond from frame to frame and apply the same technique for determining external velocities.

4.3.2 Shape Feedback

This feedback force only concerns the differences between the target boundary and the liquid boundary. However, the effect of such a force field can be easily diminished by the projection step [28, 11] which forces the entire velocity field to be divergence-free. That means the force field needs to be divergence-free to survive the projection step. To obtain such a force field, we take the following steps. We first initialize the forces on the boundary. Further optimization is performed to guarantee that the flux of the forces on the boundary is zero, which is a necessary condition for the liquid to be volume-preserving. At the end, a divergence-free force field at the interior of the liquid can be solved by considering the forces on the boundary as the boundary condition.

Force Initialization on the Liquid Boundary.

Let us focus on points on the liquid boundary. As shown in Fig. 4.2, when such a point $\mathbf{x}$ is outside the target shape, the initial force at this point is formulated as

$$\tilde{\mathbf{f}}(\mathbf{x}) = -\alpha\, d_T \frac{\nabla d_T}{\|\nabla d_T\|}; \tag{4.3}$$

otherwise, the initial force is formulated as

$$\tilde{\mathbf{f}}(\mathbf{x}) = -\alpha\, d_T \frac{\nabla d_L}{\|\nabla d_L\|}, \tag{4.4}$$

where α is the gain for proportional control. Note that the force follows the opposite direction of ∇d_T when the point is outside the target so that the liquid can be pulled toward the target faster. For the portion of the liquid surface that is inside the target shape, faster expansion is achieved by using the normal direction of the liquid surface. In our experiments, we usually set $\alpha = 625$.

Force Optimization on the Liquid Boundary.

On a discrete grid, let $\{\mathbf{n}_i\}_1^m$ be the normals of the m surface faces on the liquid boundary, and $\{\mathbf{f}_i\}_1^m$ be the finalized feedback forces on the boundary. To ensure that the total flux of these forces is zero, we enforce the following compatibility condition,

$$\Phi_{\mathbf{f}} = \sum_i \mathbf{f}_i \cdot \mathbf{n}_i = \sum_i \sum_{j=1}^{3} f_i^j\ n_i^j = 0, \tag{4.5}$$

where $\mathbf{f}_i = [f_i^1 \;\; f_i^2 \;\; f_i^3]^T$ and $\mathbf{n}_i = [n_i^1 \;\; n_i^2 \;\; n_i^3]^T$. The area of the faces has been eliminated from (4.5) since it is the same for all the grid cells.

Let $\{\tilde{\mathbf{f}}_i\}_1^m$ represent the initial forces on the liquid boundary. We would like to adjust such an initialization by minimizing the following objective function while satisfying (4.5),

$$\sum_i \|\mathbf{f}_i - \tilde{\mathbf{f}}_i\|^2, \tag{4.6}$$

where the squared differences measure the deviation from the initialization. Such a constrained minimization problem can be solved by introducing a Lagrange multiplier for (4.5). Suppose $\Phi_{\tilde{\mathbf{f}}} = \sum_i \tilde{\mathbf{f}}_i \cdot \mathbf{n}_i$. The final

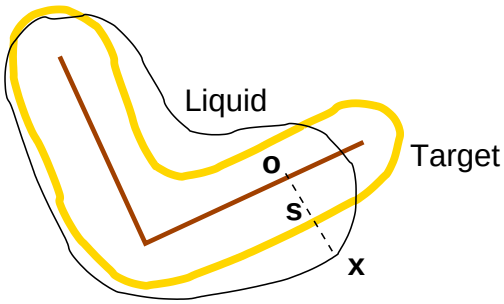


Figure 4.3: A potential field is defined with respect to the target object's shape and skeleton. The skeleton is assigned a distance value d_0 while the surface of the target is assigned a distance value d_1 with $d_0 < d_1$. The potential at any point $\mathbf{x}$ within the liquid volume is defined according to (4.10) and (4.9).

solution of the boundary forces is simply as follows [56].

$$\mathbf{f}_i = \tilde{\mathbf{f}}_i - \frac{\Phi_{\tilde{\mathbf{f}}}}{m}\,\mathbf{n}_i, i = 1, \ldots, m. \tag{4.7}$$

The Complete Shape Feedback Force Field.

Once the flux of the forces on the liquid boundary is zero, it becomes feasible to solve a divergence-free force field throughout the liquid volume. We formulate the complete shape feedback force field as the gradient field of a scalar function, H. Thus, $\mathbf{f}_{shape} = \nabla H$.

To ensure zero divergence everywhere, $\nabla \cdot \nabla H$ has to be zero everywhere in the liquid. Thus, H is a harmonic function which is the solution to the following Laplace equation with a boundary condition specified by the gradients on the boundary [56],

$$\nabla^2 H = 0, \quad \nabla H|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{4.8}$$

where $\partial\Omega$ denotes the liquid boundary, and f^* represents the boundary control forces obtained from the previous section. The discretization of (4.8) on a volume grid gives rise to a sparse linear system which can be efficiently solved by a preconditioned conjugate gradient method [31].

4.4 Adaptive Geometric Potential

Controlling smoke simulation with the help of a potential field defined by the geometric shape of a target object is first investigated in [5]. In their method, the interior of the target object is assigned with a uniform minimal potential value, and points outside the target have higher potential values. The negative gradient of the potential field, which is zero inside and nonzero outside the target, served as the driving force. The

smoke converges to the target shape when the equilibrium with minimal potential energy has been reached.

In this work, we generalize this method to liquids and design an effective potential field for them. The general form of the potential field we adopt is simply a monotonically increasing function of the signed distance of the target shape,

$$\phi(\mathbf{x}) = C \operatorname{sgn}(d_T(\mathbf{x})) \, |d_T(\mathbf{x})|^\gamma \tag{4.9}$$

where C is a constant factor representing the overall strength of the potential field. $C = 1300$ and $\gamma = 2$ in most of our experiments. The reason for this formulation is that the gradient of this new potential field has increasing magnitude when the distance to the target surface becomes larger. That means the potential field generates a larger force to pull the liquid back when it is further away, but the force remains small around the target surface to produce more natural fluid motion. This is in contrast to the original signed distance which only has a constant gradient. This new potential still has the same equilibrium as before. When the target object is moving, we simply calculate its potential field at every frame.

The negative gradient of the above potential field points towards the central skeleton of the target shape. However, using a function of the signed distance as the potential field creates problems when the target has a complex shape, such as the shape of a human character whose limbs and torso have very different thickness. Thick regions can tolerate more deviation in shape than thin regions which thus demand larger gradient forces to produce faster response. For this purpose, we define an adaptive signed distance based on two isosurfaces assuming that a connected skeleton of the target shape is known. In this definition, all the points on the surface of the target shape are required to have the same distance value, d_1, and all the points on the skeleton of the target shape are required to have another distance value, d_0 ($< d_1$). As shown in Fig. 4.3, the signed distance at a point $\mathbf{x}$ inside the liquid volume is defined to be

$$d_T^a(\mathbf{x}) = d_0 + (d_1 - d_0)\frac{\|\mathbf{ox}\|}{\|\mathbf{os}\|}, \tag{4.10}$$

where $\mathbf{o}$ is the closest point of $\mathbf{x}$ on the skeleton, and $\mathbf{s}$ is the intersection between the boundary of the target shape and the line defined by $\mathbf{ox}$. Thus, the potential field in (4.9) should be updated by replacing $d_T(\mathbf{x})$ with $d_T^a(\mathbf{x})$. With such an adaptive potential field, thin regions have a larger gradient force because the Euclidean distance between the skeleton and the surface is smaller. Another advantage of considering the skeleton as an isosurface is that it is easier for the liquid to move around and less likely to be trapped in a local region. In practice, we always set $d_0 = -1$ and $d_1 = 0$. In general, such a gradient force field does not guarantee zero divergence.

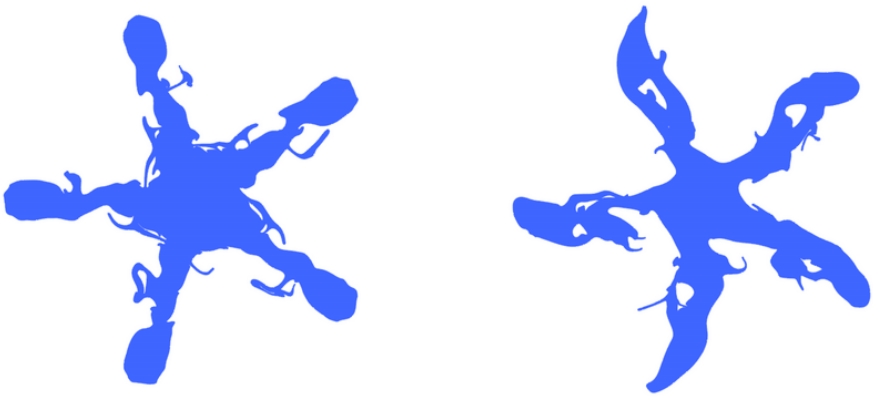


Figure 4.4: Two frames from a two-dimensional controlled fluid simulation with a rotating star shape as the target motion. The 2D grid resolution is 1000^2, and the computational time is 2.5 minutes per frame.

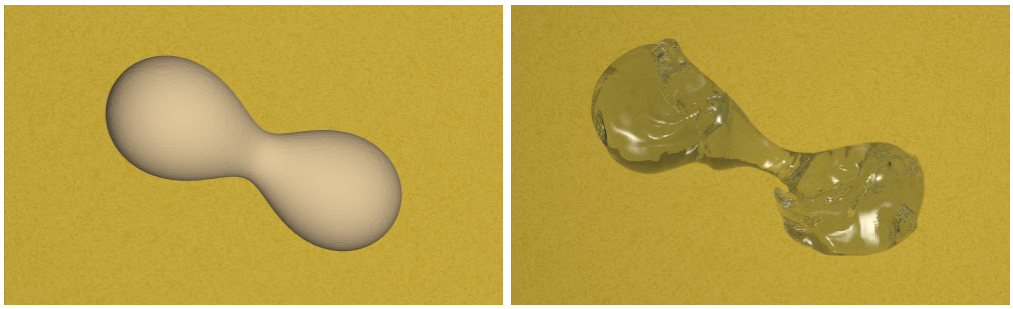


Figure 4.5: Left: The target shape and orientation at one of the frames of the target animation. Right: A rendered image from our liquid control method. The resolution is 300^3, and the computational time is 3.2 minutes per frame.

4.5 Implementation and Analysis

The input to our system is a continuous animation of single or multiple target shapes, which can be created from key-frames, physics-based simulations, or motion capture. The sequence of animated target shapes should be automatically converted to a sequence of implicit functions because the liquid surface is represented as an implicit function as well. The output from our system is a liquid simulation that approximately follows the target animation while still maintaining natural liquid dynamics.

Our overall control scheme exploits both types of forces introduced in the previous two sections. The feedback control force field is very powerful in making the liquid follow the target shape and motion. However, such a strong control is not desirable because the liquid motion would not look natural anymore. On the other hand, the force generated by the geometric potential field resembles the gravity in the real world. If we look at the earth from the space, the gravity actually points roughly towards the center of the earth.

Due to this similarity, when the target does not have large accelerations, the controlled fluid motion under the geometric potential is expected to be similar to that under the gravity, and therefore, appears natural. However, the control capability of the geometric potential weakens and produces undesirable high frequency oscillations when it is changing quickly from frame to frame. The velocity feedback force can effectively alleviate such oscillations. Therefore, to achieve both desirable control and natural fluid-like motion, we apply both force fields simultaneously, but use a relatively small feedback force to avoid artifacts. The typical overall strengths of these force fields have been given in the previous sections.

According to Section 4.3 and 4.4, our control technique computes spatially varying force fields automatically. Such computations are fast, and do not incur much extra cost beyond a regular fluid simulation. As a result, we can adopt any grid resolution that a regular fluid simulation can use. There are only three parameters that an animator needs to interactively adjust to achieve desirable controlled simulation. Namely, the gain of the shape feedback, the gain of the velocity feedback and the overall strength of the geometric potential field. It should be noted that a small amount of user intervention is necessary for an authoring tool since an animator should be able to choose preferable results by tuning a small number of "control knobs". In our experiments, the control parameters do not need to be changed drastically for different target sequences. An animator should be able to author new animations by incorporating minor changes to existing parameters. One can also start with relatively strong control parameters, and relax them until the results become desirable. Experiments also indicate that our force fields are insensitive to the resolution of the simulation grid. Similar effects can be obtained at various different resolutions except for very coarse ones. Thus, a coarser grid can be used during the tuning stage to let the animator quickly see intermediate results. Therefore, it typically only takes a few hours to author a reasonable fluid animation with controlled behavior.

4.6 Experimental Results

We have successfully applied our liquid control method to a few examples. Fig. 4.4 shows two frames from a two-dimensional controlled fluid simulation which uses a rotating star shape as the target animation. The overall shape of the fluid boundary resembles the target object. However, there are also many intricate details indicating this is a fluid. Fig. 4.5 shows a three-dimensional controlled simulation which uses the rigid body motion of a dumbbell shape as the target animation. The liquid not only follows the target motion, but also has small waves on the surface.

The left image in Fig. 4.6 shows one frame from a controlled fluid simulation with a dolphin jumping

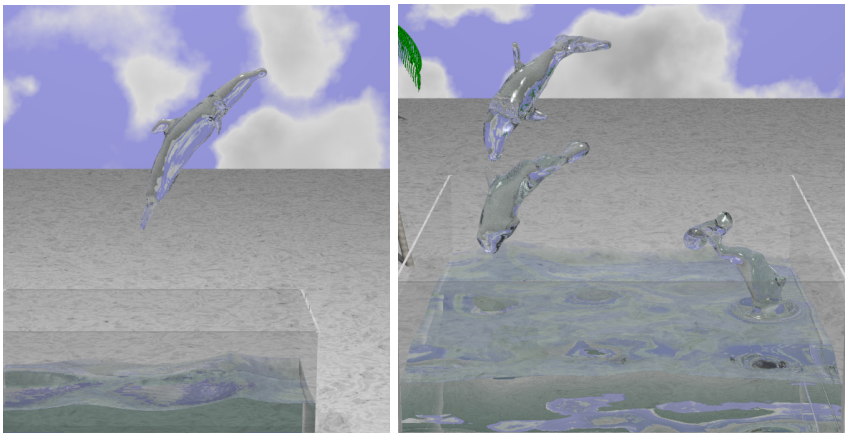


Figure 4.6: Left: One liquid dolphin in the air. The resolution is 234x180x108, and computational time is around 4.8 minutes per frame. Right: Three liquid dolphins jumping back to the water. The resolution is 180x78x180, and computational time is 4.2 minutes per frame.

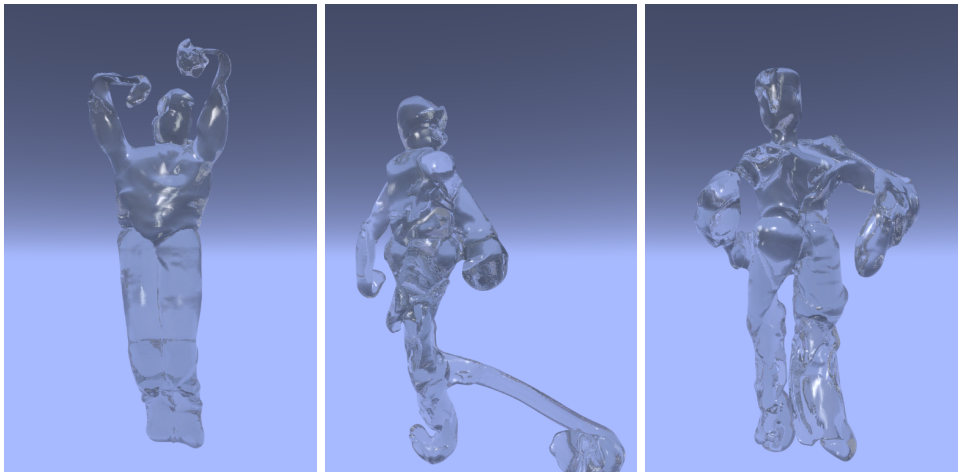


Figure 4.7: A liquid simulation following a dancing sequence. The simulation grid resolution is 300^3, and the computational time is 7 minutes per frame.

sequence as the target animation. When the target jumps out of the water, it also brings a portion of the water with it. The isolated portion of the water evolves into a dolphin shape. The right image in Fig. 4.6 shows another related simulation with three dolphin-shaped targets.

Fig. 4.1 and 4.8 show a liquid horse emerging from a water, running on the water and then collapsing back into the water. Both the simulated liquid surface (blue meshes) and the underlying target shape (yellow meshes) are shown as well. The target shape has two components, the horse shape and a horizontal plane under the horse. The horizontal plane defines the target shape for the free moving water which is supposed to have a flat surface when it stabilizes. The horse mesh sequence is from [57]. Fig. 4.8(a)-(c) show interesting liquid dynamics when the shape of the horse is being formed. Fig. 4.8(d) shows a frame where the liquid has stabilized and matched the target shape. Most small-scale details on the target shape are preserved by the

matching liquid interface. To generate the collapse in Fig. 4.8(e)-(f), we simply remove the horse shape and only leave the horizontal plane. Interestingly, we obtained more natural results in this example by setting $\gamma = 1$ in (4.9).

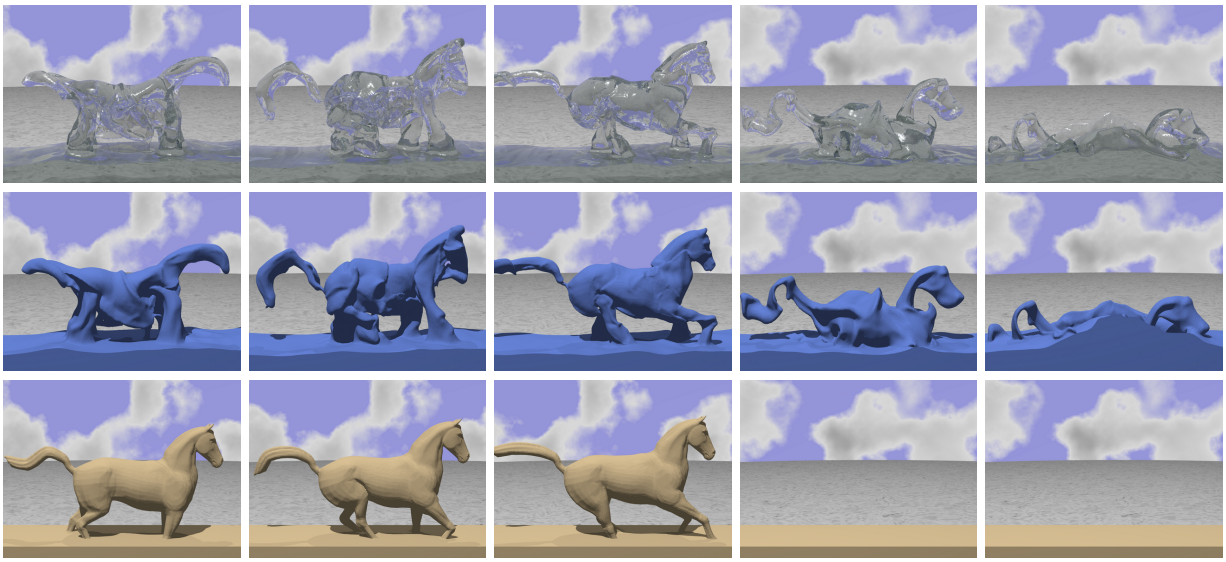
Fig. 4.7 shows another controlled liquid simulation using a MOCAP sequence as the target animation. The MOCAP data have very large, rapidly changing accelerations. Nevertheless, our control method was able to keep the liquid in pace with the target. Meanwhile, the simulated liquid surface exhibits natural waves and other fluid motion. A comparison is given in Fig. 4.9. We tested our method on this sequence against the original potential-based method in [5] and the smoke control technique in [6]. To preserve fluid appearance, a weak potential field is used in Fig. 4.9(b), but the liquid does not follow the rapidly changing target shape. Once a relatively strong potential field is used to control the liquid as in Fig. 4.9(c), the liquid does not have natural motion but many high frequency oscillations. In Fig. 4.9(d) we show the method from [6] cannot control the liquid well either even with an extremely strong force field. All the animations can be found in the accompanying video.

In our experiments, we use an effective grid resolution up to 300^3 using a grid windowing and resizing technique similar to that in [45]. The computational time is almost the same as a regular liquid simulation which costs 5-10 minutes a frame on a high-resolution grid and only around 30 seconds a frame on a coarse grid ($< 100^3$) using an AMD 3200+ processor. Computing the control force fields only costs less than 10% of the total simulation time. It should be noted that the finest grid resolution used in [16] is 50x50x50 which only has 125,000 elements. Thus, the grid in their simulations has two orders of magnitude less elements than ours.

4.7 Conclusions and Future Work

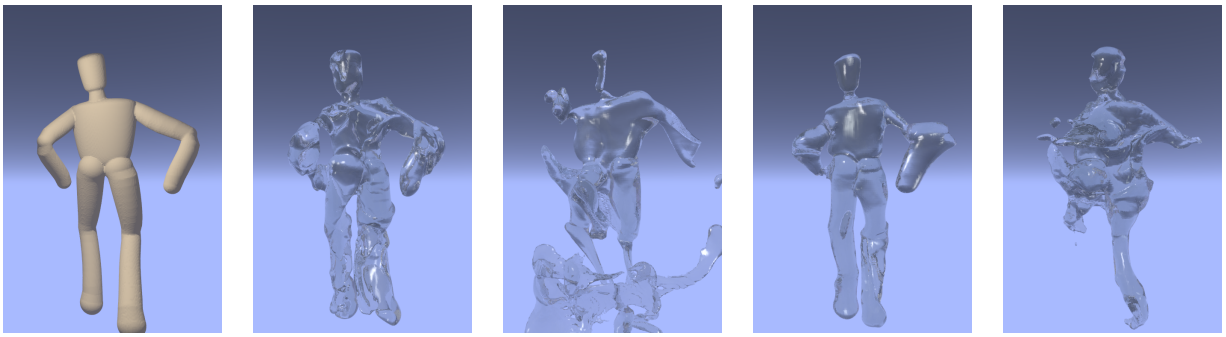
Following rapidly changing target objects is a challenging problem in fluid control. We introduced a simple but very effective solution by applying two external force fields: a carefully designed feedback force field and the negative gradient field of a geometric potential function. Experiments indicate that our method can achieve pleasing results.

In future, we would like to extend this work to sparse keyframes and include interactions between the controlled liquid and other more rigid objects. During such interaction, interesting phenomena such as splashing should be simulated. A hybrid system based on both target shapes and control particles might be necessary for such simulations. The technique in [50] might also be helpful.



(a) (b) (c) (d) (e)

Figure 4.8: First row: a liquid simulation representing a water horse emerging (a & b), running (c) and collapsing (d & e). Second row: the simulated liquid surfaces rendered as meshes. Third row: the underlying target animation. The resolution is 275x250x75, and the computational time is 4.4 minutes per frame.



(a) (b) (c) (d) (e)

Figure 4.9: (a) The target shape at one of the frames of the dancing sequence. (b) A rendered image from our liquid control method. (c) A rendered image from the method in [5], with a weak potential field. (d) Same method as in (c) with a relatively strong potential field. (e) A rendered image from the method in [6].

Chapter 5

Controllable Motion Synthesis in A Gaseous Medium

5.1 Introduction

The generation of natural motion that satisfies user-defined requirements is an important goal of computer animation. This article presents a technique to synthesize realistic, controllable motion for lightweight objects, alone or in a group, being acted on by a gaseous medium. The techniques we present are applicable in synthetic animation and video production because lightweight objects, such as leaves, feathers and bubbles, are ubiquitous in our environment, and their motion suggests grace and fluidity.

Real world moving objects receive influences (external forces) from the surrounding air, the most apparent being air drag. At the same time, air has its own dynamics and receives influences from the objects. We define "lightweight" objects to be those that do not strongly influence the large scale behavior of the air, so that the motion of one object does not significantly impact the motion of its neighbors. Furthermore, lightweight objects largely follow the velocity field of the air, and hence their motion provides a visualization of most characteristics of the underlying flow. In particular, most gaseous phenomena should be almost divergence free, and hence the lightweight objects should rarely, if ever, collide with each other as they approximately follow streamlines.

We focus in on the motion of the objects themselves, because they are directly visible. We are concerned with the fluid only to the extent that it can be interpreted from the objects' motion. Hence we frame the problem as one of object motion synthesis in the presence of constraints imposed by the fluid. Inspired by the seminal work on spacetime constraints [22], we formulate this problem as a large-scale optimization. Directly solving this large-scale problem using an off-the-shelf optimization toolbox would be infeasible since the number of parameters necessary for realistically describing fluid motion is prohibitively large, especially when the fluid is simulated using a voxel grid. Therefore, we have devised a number of novel and effective methods to make this problem tractable.

• *Smoothed particle hydrodynamics* (SPH) are used [58, 59, 32] to describe fluid motion and provide optimization constraints. Although SPH is numerically less accurate than a grid-based simulation, it can

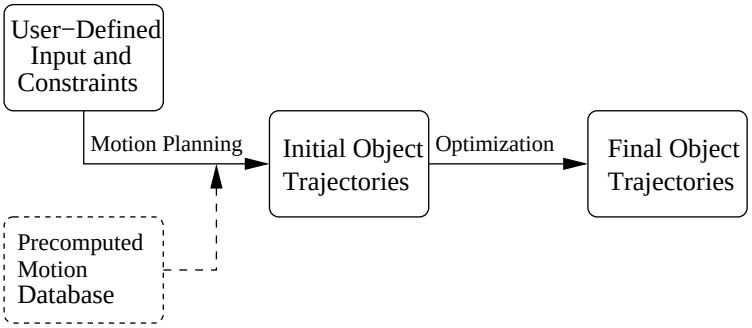


Figure 5.1: System overview.

generate reasonable fluid behavior using a relatively small number of particles which significantly reduces the number of constraints required in the optimization. The reduction in accuracy is acceptable because we do not directly visualize the fluid, but the objects immersed in the fluid.

• *Data-driven synthesis* is exploited to generate an initial trajectory for each object. The trajectory may have certain desirable features required by the user. The optimization then makes the initial trajectories more physically plausible. The user is thus able to guide the optimization toward a specific local minima by biasing the starting point. Such control is necessary because the problem may have numerous equally plausible local minima, while a user almost certainly has a specific one in mind.

• *Vortexes* are important in producing interesting fluid motion. During optimization, vortices's are identified and recovered in a separate pass to improve efficiency.

• *Rotational components* of the objects are largely independent of their overall trajectory. We ignore them in the optimization and refine them in a separate post-processing stage to produce visually pleasing results.

5.2 Overview

A dataflow diagram of our system is given in Fig. 5.1. The *input* to our problem includes a set of lightweight rigid objects with six DOFs, their initial configurations (mass, positions, velocities and etc.), a set of user-specified constraints (positions, velocities and even accelerations of certain objects at specific times), and the environmental configuration including a set of obstacles. The *output* includes the complete specifications of the DOFs of the objects at every frame of an animation. In addition, the objects should have plausible motion in a gaseous medium, reflecting characteristics of the underlying fluid motion. In particular, the object motion should appear to be divergence free and continuous. The user controls the final result via the set of constraints. There are two convenient ways to specify the constraints, key frames and partial

trajectories. A keyframe specifies the positions of all the objects at a particular time simultaneously while a partial trajectory specifies a subsequence of the positions of a single object.

In this problem, the objects have passive motion largely controlled by the gas. To reflect interesting motion of the fluid, we need a reasonably accurate CFD algorithm while the objects can be modeled in a simple manner. We use particles as the dynamic model for both the gaseous medium and the objects. The gas consists of a number of particles coupled together using smoothed particle hydrodynamics (Section 2.3). Each object is approximated with a small number of particles whose relative positions are fixed throughout the animation. Gas particles and object particles are labeled differently, and have different mass. Gas particles also interact with object particles following SPH, while different objects do not directly interact with each other.

We cast this problem as a space-time constrained optimization which is the last stage in Fig. 5.1. Since the gas and object particles have interactions, the unknowns include the positions and velocities of all the particles at all frames. The SPH equations of motion as well as user-specified controls are set up as optimization constraints. Directly optimizing such a large-scale problem without a reasonable initialization would be infeasible. We actually devise a data-driven motion synthesis algorithm to generate initial trajectories of the objects before nonlinear optimization. The initial trajectories are synthesized using a method similar to those in Lee et al. [60], Kovar et al. [61] and Arikan et al. [62]. However, our synthesis method is based on robotic motion planning [63, 64] and therefore, can avoid obstacles as well as satisfy position constraints. Since such a synthesis requires a large motion database to start with, we synthetically run gaseous simulations using the method of Fedkiw et al. [29] with a number of combinations of force fields and obtain a space-time dataset from each simulation. We trace motion trajectories from all simulations to form the database. This database is precomputed only once and used to generate initial trajectories for all examples in this chapter.

We also model forces explicitly since they are responsible for interesting motion. All the forces together are modeled as a force field which is further decomposed into two components: a vortex force field with zero divergence but nonzero curl, and a laminar field with smooth force vectors. Every trajectory from the motion database is annotated with the type of force field that generated it during gas simulations. During optimization, if a segment of an initial trajectory is annotated with a vortex field, a parametric vortex model is fit to the forces along the trajectory first. We carry out vortex fitting and the aforementioned space-time optimization in two alternating passes.

5.3 Pre-computing the Trajectory Database

In this section, we discuss the steps in building the motion database for subsequent motion planning. Since this part is not the focus of this work, we only present the outline of the steps and refer the readers to the relevant papers for further details.

5.3.1 Fluid Simulation

Building the database requires a large number of sample trajectories. Since capturing trajectories from the real world seems infeasible and fluid simulation has become quite mature recently, we collect the sample trajectories from grid-based gaseous simulations using the algorithm of Fedkiw et al. [29], which can produce high quality results. This overall approach is similar to the pre-computation stage in James and Fatahalian [65] which focuses on deformable objects.

We use a 64^3 grid with the Neumann boundary condition. We run a number of simulations each of which has a distinct force field and an optional set of simple obstacles, such as boxes and cylinders. Each force field is superposed from multiple vortex fields and wind fields. The vortex fields in 3D are defined parametrically using a pair of poles $\mathbf{p_1}$ and $\mathbf{p_2}$ as follows [66].

$$\mathbf{f}_{vtx3d}(\mathbf{x}) = \omega \frac{(\mathbf{r_0} \cdot \mathbf{r_1}/\|\mathbf{r_1}\| - \mathbf{r_0} \cdot \mathbf{r_2}/\|\mathbf{r_2}\|)}{4\pi\|\mathbf{r_1} \times \mathbf{r_2}\|^2} \mathbf{r_1} \times \mathbf{r_2} \tag{5.1}$$

where $\mathbf{r_0} = \mathbf{p_2} - \mathbf{p_1}$, $\mathbf{r_1} = \mathbf{x} - \mathbf{p_1}$, $\mathbf{r_2} = \mathbf{x} - \mathbf{p_2}$, and ω is the strength of the vortex. We define a space-time vortex as $\mathbf{f}_{vtx}(\mathbf{x}, t) = G(t)\mathbf{f}_{vtx3d}(\mathbf{x})$, where $G(t)$ is a Gaussian in the temporal space. And the wind fields have parallel force vectors with a deteriorating strength in both space and time. We start tracing a trajectory from the center of a voxel in the first frame of a simulation, and keep tracking the trajectory from frame to frame by following the velocity vectors until the last frame or the boundary of the work space has been reached. Each point on the trajectory is annotated with the type of force field present in its neighborhood. Since the simulation grid has an unnecessarily large number of voxels, we only trace a trajectory from a subset of the voxels uniformly distributed in the simulation grid. According to our experiments, the degree of freedom in the shape of the trajectories is not overly large. We stop observing new trajectory shapes once the number of simulations exceeds a certain threshold, or the length of the simulations exceeds a certain duration. Thus, we empirically decide the number of simulations we need and the duration of the simulations.

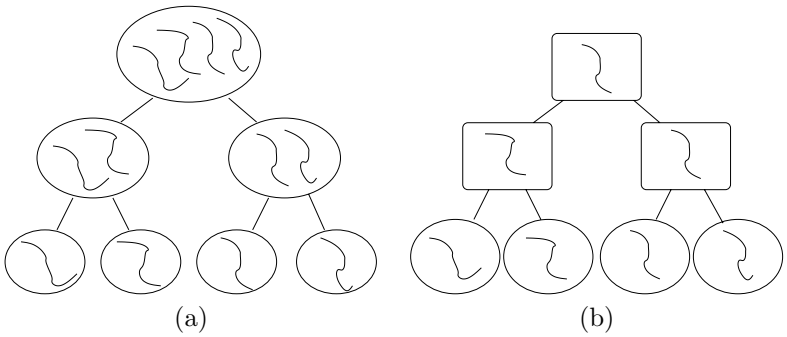


Figure 5.2: (a) An initial cluster of trajectory segments is recursively split into smaller ones until they only have one segment each; (b) There is a tree structure corresponding to this recursive splitting process. Each node in the tree contains a representative of the segments in its subtree. The segments in the original cluster are distributed among the leaf nodes.

5.3.2 Compression and Clustering

The amount of raw trajectory data is too large to be conveniently used for any subsequent steps. Thus we need to perform segmentation, compression and clustering. Segmentation partitions each complete trajectory into shorter segments which are considered as motion units that can be rejoined during motion synthesis. We compute a curvature at every point on the trajectory and classify high curvature portions of the trajectory as features. We maintain the integrity of the features by partitioning the trajectory at points with locally minimal curvature. The segments thus obtained have various different lengths. Segments with approximately the same length are grouped together, and truncated to have exactly the same length. Further compression and clustering are performed separately on each group. We use singular value decomposition (SVD) to compress the segments as in James and Fatahalian [65]. Note that SVD does not reduce the number of segments, but reduces the dimensionality required to represent each segment.

In the next step, tree-structured vector quantization (hierarchical k-means clustering) [67] is performed to organize the segments from each group into tree structures which are very efficient for nearest neighbor search. Each group starts from a small number of clusters obtained from an initial k-means clustering. Each of the initial clusters is recursively split into smaller ones which have corresponding nodes in an accompanying tree structure (Fig. 5.2). An internal node of the tree contains a representative of the segments in its subtree.

5.4 Stylistic Motion Planning

One of the goals of robotic motion planning is to search for a path between a source and a destination while avoiding obstacles. By *stylistic motion planning*, we mean that in addition to path searching, the trajectory and the velocities along the trajectory should mimic the motion style of a class of natural objects.

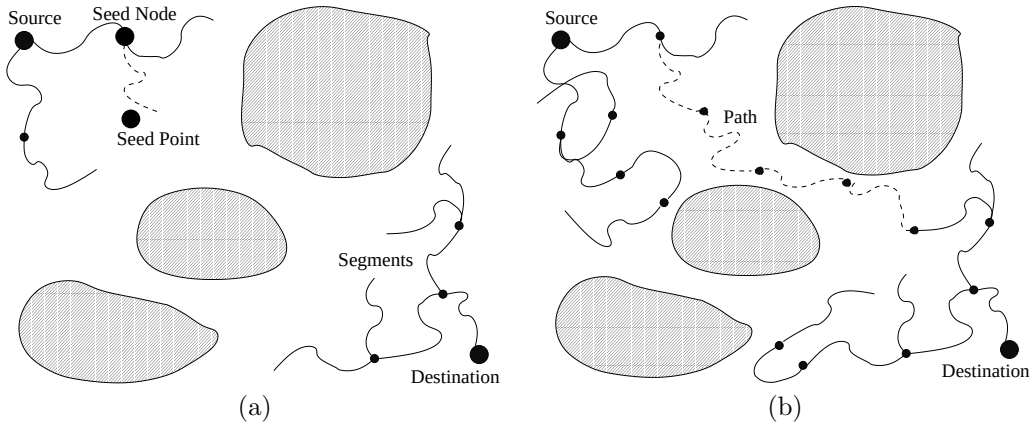


Figure 5.3: Path planning by growing two probabilistic trees from the source and destination, respectively. (a) A new segment (dashed branch) is inserted into the source tree given a seed point. (b) A path (dashed) is found when two branches from the two trees become connected.

For example, suppose we would like to move a synthetic feather from a source to a destination in the air, stylistic motion planning should find a motion that not only satisfies the position constraints, but also makes the fake feather floating and tumbling like a real one.

We designed a technique to achieve stylistic motion planning by generalizing a probabilistic robotic motion planning algorithm called Rapidly-Exploring Random Trees (RRTs) [63, 64]. The idea is to adapt this algorithm to assemble a path only using trajectory segments from the precomputed motion database. The trajectory segments represent motion units with a certain desired style.

Motion planning based on RRTs does bidirectional path planning by extending two random trees from the source and destination, respectively. During the planning, new path segments avoiding obstacles are appended to the leaves or internal nodes of the two trees in a probabilistic manner so that both trees keep growing until a segment from one tree comes into contact with a segment from the other, which indicates a path has been found. This technique can achieve better performance than other existing path finding algorithms, including probabilistic roadmaps used by Choi et al. [68]. In the original RRTs, all the path segments have the same shape, such as a line segment or a circular arc, and there are no continuity requirements between consecutive segments in a path.

In our stylistic planning, every branch on the two trees has to be a segment sampled from the motion database. Adjacent segments in the trees are likely to be different. In addition, consecutive branches should maintain a certain degree of continuity to make the final motion look natural. In order to be able to evaluate the smoothness of the connection between two consecutive segments as well as facilitate smooth blending between them, we enforce an overlapping portion between them.

To extend a new segment from one tree, we randomly generate a seed point in the search space and

identify the closest node from the tree as the *seed node* (Fig. 5.3). We search the database to sample a segment that satisfies two conditions: i) the segment can be smoothly connected to the parent branch at the seed node after a rigid body transformation (Note that the path may become unphysical once a rigid body transformation has been applied. However, the subsequent optimization introduced in the next section can remove the artifacts); ii) once one endpoint of the segment is attached to the seed node, the other endpoint is closest to the seed point (Fig. 5.3(a)). If the sampled segment hits any obstacle (collision detection) once in position, we simply discard the segment and regenerate a seed point. Otherwise, we compute the minimum connection cost between the new branch and any branch in the other tree, and update the minimum connection cost between the two trees. The connection cost between two branches measures the total amount of translation and rotation necessary for a smooth connection, and will be formally defined in Equation 5.2. The planning process stops whenever the minimum connection cost drops below a threshold, or a maximum number of tree branches have been grown.

In the next stage, we form a path between the source and destination by joining the two tree branches (one from each tree) that actually achieve the minimum connection cost between the two trees (Fig. 5.3(b)). Obviously, this last connection introduces the worst continuity in the whole path. Therefore, we uniformly redistribute the discontinuity to all connections in the path. For efficiency, so far we have been using the representative segments at the top level of the tree-structured clusters in the database. Next, we refine the continuity at all connections by descending into lower levels of the clusters. Since every segment s_i in the path except the first and the last has two connections with its preceding and following segments, we traverse the subtree rooted at s_i in the database to potentially find another segment that gives the optimal continuity at these two connections. Finally, we blend every two consecutive segments in their overlapping portion.

To significantly improve performance, we pre-compute a lookup table that records the connection cost between any pair of representative segments at the top level of the tree-structured clusters. This computation only considers the overlapping portion of the segments(the tail window of the first segment and the head window of the second segment), and find a translation/rotation matrix $\mathbf{M}$, such that those two windows will have best matching. This is equivalent to the following minimization,

$$\text{argmin}_{\mathbf{M}} \sum_j \|\mathbf{x}_1^j - \mathbf{M}\mathbf{x}_2^j\|^2 \tag{5.2}$$

where $\mathbf{x}_i^j$'s represent the points in the two windows. Note that the L^2 norm can be replaced by other measures. An approximate solution for the matrix $\mathbf{M}$ can be found by solving the principal axes of the two windows using principal component analysis (PCA) and aligning the two local frames defined by the two

sets of principal axes. At a smaller computational cost, this approximate solution provides very good results in practice comparing to true optimization. If the connection cost estimated using the matrix $\mathbf{M}$ is larger than a threshold, we mark the corresponding entry of the lookup table as "not-connectible", otherwise we record the matrix in the table.

5.5 Optimization

We resort to a large-scale space-time optimization at the end to improve the initial dynamic appearance produced by motion planning as well as better enforce additional constraints other than position constraints. The motivation is twofold. Certain transitions in motion planning may not be physically realizable since the segments were simply concatenated and blended together. Furthermore, spatial coherence due to fluid motion was not enforced since the trajectories for multiple objects were planned independently. Although we follow smoothed particle hydrodynamics (Section 2.3) in this section, the basic ideas of optimization can be presented relatively independent of the details in the SPH formulations. In the remainder of this section, we are going to focus on these basic ideas.

The optimization searches for a plausible fluid dynamic simulation that satisfies all user-defined requirements. Since the objects and gas are coupled together with interactions, we need to define a work space filled with gas particles in addition to the object particles even though the final solution we seek only concerns the objects. We choose to consider the state variables (positions) of all the particles and their derivatives as the set of unknowns. Thus, the actual SPH formulations are enforced as additional optimization constraints. The initial trajectories of the object particles are obtained from path planning while the gas particles are initially positioned on a regular grid with zero velocities.

In addition to the state variables of the particles and their derivatives, the force field also needs optimization. We employ a hybrid representation of the force field, parametric vortex fields plus a nonparametric wind field which is required to have smooth direction and magnitude throughout the 4D space-time. A vortex is basically a rotational vector field with a field strength deteriorating with distance and time. A parametric vortex model is a 4D function of space and time, and is denoted as $\mathbf{f}_{vtx}(\mathbf{x}, t)$.

Let us denote the state of all the particles as $\mathbf{q} = \{\mathbf{x}_i\}_{i=0}^{n-1}$, and its derivative as $\dot{\mathbf{q}} = \{\mathbf{v}_i\}_{i=0}^{n-1}$. For the j^{th} frame, we denote the state variables and their derivatives as $\mathbf{q}_j = \{\mathbf{x}_i^j\}_{i=0}^{n-1}$ and $\dot{\mathbf{q}}_j = \{\mathbf{v}_i^j\}_{i=0}^{n-1}$, respectively, where $\mathbf{x}_i^j$ and $\mathbf{v}_i^j$ indicate the position and velocity, respectively, of the i^{th} particle in the j^{th} frame. Thus, the complete set of variables that need optimization is represented as $S = \{\mathbf{q}_j, \dot{\mathbf{q}}_j, \{\mathbf{F}_i^j\}_{i=0}^{n-1} | 0 \leq j < f\}$, where $\mathbf{F}_i^j$ represents the force vector at $\mathbf{x}_i^j$. Note that $\mathbf{F}_i^j = \mathbf{F}_{vtx4d}(\mathbf{x}_i^j, j) + \mathbf{h}_i^j + m_i g$ where $\mathbf{F}_{vtx4d}(\mathbf{x}_i^j, j)$

represents the accumulated vortex force at $\mathbf{x}_i^j$, $\mathbf{h}_i^j$ represents the wind force at $\mathbf{x}_i^j$ and $m_i g$ is the gravity. In the following, we actually only consider the vortex component of $\mathbf{F}_i^j$ as a free variable while the wind component is implicitly constrained.

Our space-time optimization is a constrained optimization with a cost function and a set of hard constraints. The user-defined requirements are treated as hard constraints. The complete set of such constraints are denoted as $C(S) = \{C_k^j | 0 \leq k < m_j, 0 \leq j < f\}$ where m_j is the number of constraints at the j^{th} frame, and C_k^j is a constraint on either $\mathbf{x}_i^j$ or $\mathbf{v}_i^j$. For example, a constraint C_k^j on $\mathbf{x}_i^j$ has the form $\|\mathbf{x}_i^j - \mathbf{o}_k\| \leq b_k$. A strict positional constraint is imposed when b_k is set to zero. A constraint on $\mathbf{v}_i^j$ can be defined similarly.

The cost function for the optimization consists of three terms:

$$E(S) = c_1 E_x(S) + c_2 E_v(S) + c_3 E_c(S) \tag{5.3}$$

where c_i's are coefficients indicating the importance of each term, E_x and E_v represent the penalty terms to enforce the SPH formulation, and E_c represents the spatio-temporal smoothness of the velocity field. To explain the details in these terms, let us consider the j^{th} frame. Given the particle positions and velocities associated with the $(j-1)^{th}$ frame, we should be able to run SPH simulation for one frame and predict these quantities for the j^{th} frame. Let us denote the predicted quantities as $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{v}}$. If the SPH formulation is strictly observed, $\mathbf{x}_i^j = \tilde{\mathbf{x}}_i^j$ and $\mathbf{v}_i^j = \tilde{\mathbf{v}}_i^j$. Otherwise, we penalize the discrepancies between them.

$$\begin{aligned} E_x(S) &= \sum_{ij} \|\tilde{\mathbf{x}}_i^j - \mathbf{x}_i^j\|^2 \\ E_v(S) &= \sum_{ij} \|\tilde{\mathbf{v}}_i^j - \mathbf{v}_i^j\|^2 \end{aligned} \tag{5.4}$$

Note that the SPH formulation is not enforced as hard constraints in the optimization simply to make it tractable because a large number of hard constraints are much more expensive to maintain than soft penalty terms.

To enforce the spatio-temporal smoothness of the velocity field, we require the velocity at $\mathbf{x}_i^j$ to be close to the average velocity of nearby particles in the same frame as well as average velocity of the same particle in the previous and subsequent frames, thus E_c has the form:

$$E_c(S) = c_{3,1} \left(\mathbf{v}_i^j - \frac{\sum_{k \in N_i^j} \mathbf{v}_k^j}{|N_i^j|} \right)^2 + c_{3,2} \left(\mathbf{v}_i^j - \frac{\sum_{k=j-w}^{j+w} \mathbf{v}_i^k}{2w} \right)^2 \tag{5.5}$$

where coefficients $c_{3,1} + c_{3,2} = 1$, N_i^j represents the index set of the neighboring particles of $\mathbf{x}_i^j$ in the j^{th}

frame, and w represents the size of the temporal window.

Compared to traditional spacetime constraints [22], our formulation introduces particle velocities as additional variables in the optimization while not directly optimizing the wind forces. This is because the SPH equations do not allow the velocities to be easily expressed as finite differences of positions. Therefore, they cannot be treated as derived variables. On the other hand, it is not necessary to optimize the wind forces any more because the smoothness of the velocity field implicitly constrains the wind forces.

We optimize the cost $E(S)$ and vortex force fields in two alternating stages, giving vortex fields higher priority because they introduce interesting fluid motion. Given an estimate of the $\mathbf{x}_i^j$ and $\mathbf{v}_i^j$, we can estimate the $\mathbf{F}_i^j$ from Equation 2.14. Meanwhile, we also inherit the annotations on the object trajectories obtained from motion planning. When a segment of the trajectories is annotated with a vortex field, we fit the parametric spatio-temporal vortex model to the estimated forces at the points along that segment. The parameters of a vortex model include the spatio-temporal center, the orientation, strength, and duration. After the vortex fitting stage, we fix the vortex fields and optimize $E(S)$ subject to constraints to update all the state variables except the user-defined ones. We alternate these two stages a few times to obtain the final solution. The optimization is carried out using sequential quadratic programming [69].

There are two additional details regarding this large-scale optimization. First, to make the initial trajectories from path planning have sufficient influence on the final solution, we allow gradually larger deviations from the initial trajectories as the optimization progresses. At the beginning, every point on the trajectories is enforced as a position constraint with gradually increasing radius. When the radius has become sufficiently large, we then remove the position constraint. Without this strategy, the initial trajectories may be destroyed fairly early in the optimization process without much influence on the final solution. Second, it may not be feasible to optimize all the variables simultaneously on a machine with a limited amount of memory. We only optimize a subset of the variables at a time with the rest of the variables fixed. The optimization is performed repeatedly on disjoint subsets of variables, but does not wait until convergence on each subset to avoid local minima.

5.6 Rotation Refinement

Since we only use a small number of particles to represent each object during the space-time optimization using SPH, we do not expect the rotational components to be very accurate. Therefore, we only obtain the translational component at the center of mass of each object by averaging the translations of all the particles of that object. More accurate rotational components with better visual effects are estimated in a

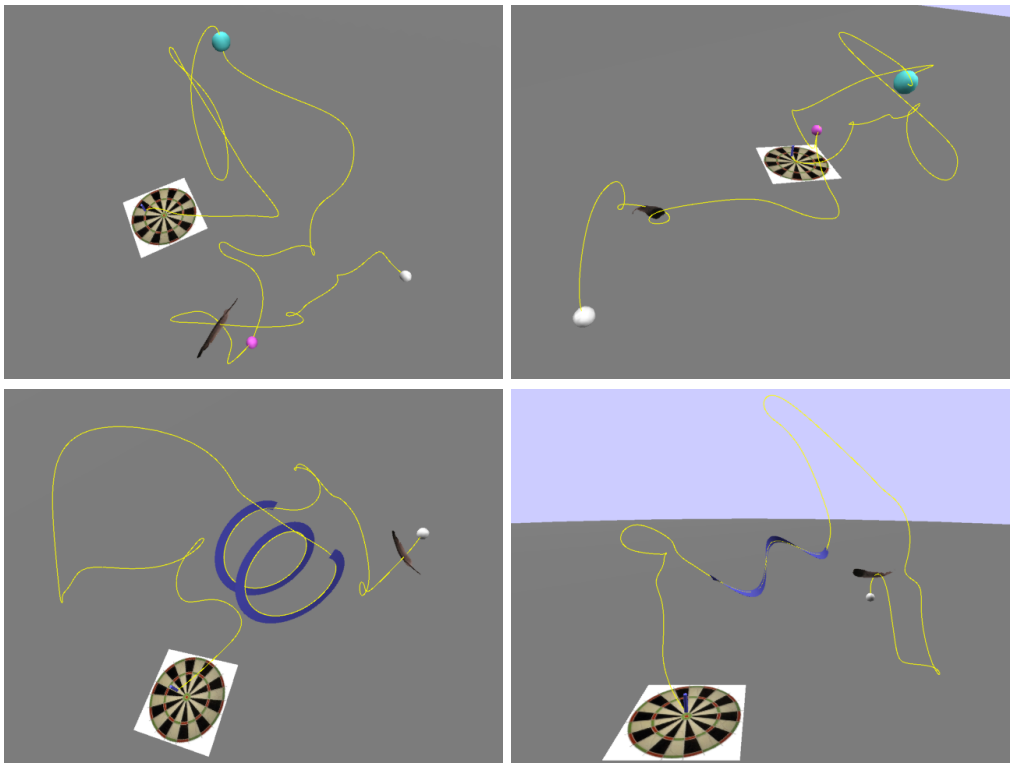


Figure 5.4: Top row: two examples of single object motion that satisfies two intermediate position constraints (red and green spheres). Bottom row: two examples of single object motion with a partially constrained trajectory (blue spirals). The white sphere is the source and the cylinder on the dart board is the destination. All trajectories are generated with path planning and optimization.

post-processing step where a polygonal model is used for each object.

The rotation of the objects in an animation is based on a simulated torque from air resistance. The force that creates the torque is a drag force that acts in the opposite direction of the velocity of the moving object. The formula for the magnitude of this drag force is

$$F_d = 0.5\, c_d\, \rho\, v^2\, A \tag{5.6}$$

where c_d is the drag constant specific to the object, ρ is the density of the gaseous medium, v is the velocity of the center of mass of the object, and A is the foreshortened area over which the force acts.

In order to approximate the torque efficiently, the shape of the object is projected onto a plane perpendicular to the direction of the drag force acting on the object. A new coordinate system within this plane is centered on the object's center of mass. The object is then rasterized, and the torque on the object is compiled by iterating across all the pixels covered by this projected object. The original torque equation simplifies to two separate ones (one for each dimension within the plane in this new coordinate system). Af-

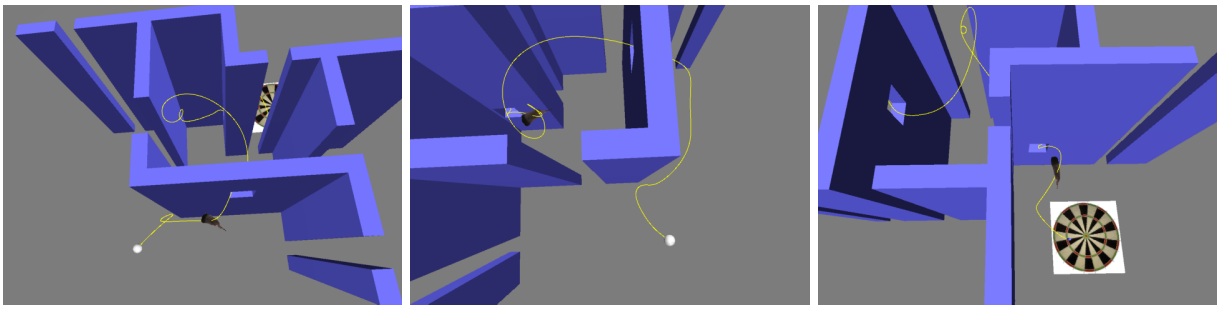


Figure 5.5: An example of stylistic path planning with obstacle (the maze) avoidance. The white sphere is the source and the cylinder on the dart board is the destination.

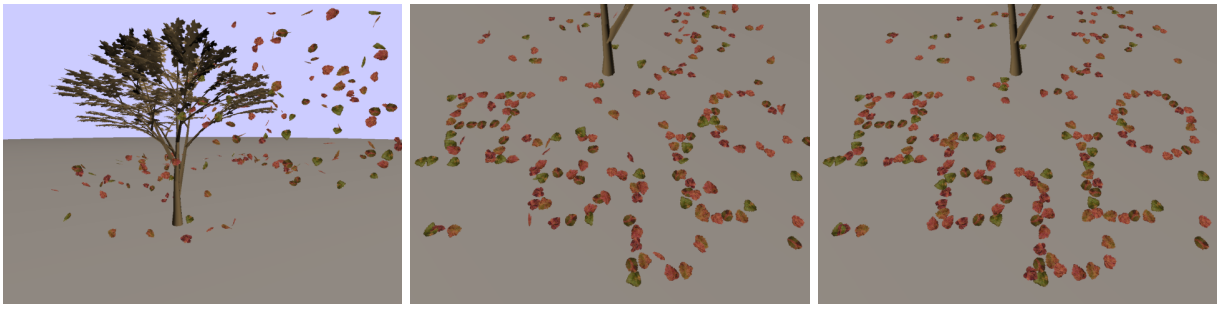


Figure 5.6: Leaves fall onto the ground, forming letters.

ter calculating and summing the torques contributed by individual pixels, the resulting accumulated torque is still specified in this new coordinate system. A simple transformation from this local frame to the global coordinate system gives the torque in terms of the global coordinates. The torque and angular velocity of the object can then be integrated in order to animate the object. In addition, damping based on the object's angular velocity has been used to simulate rotational drag forces.

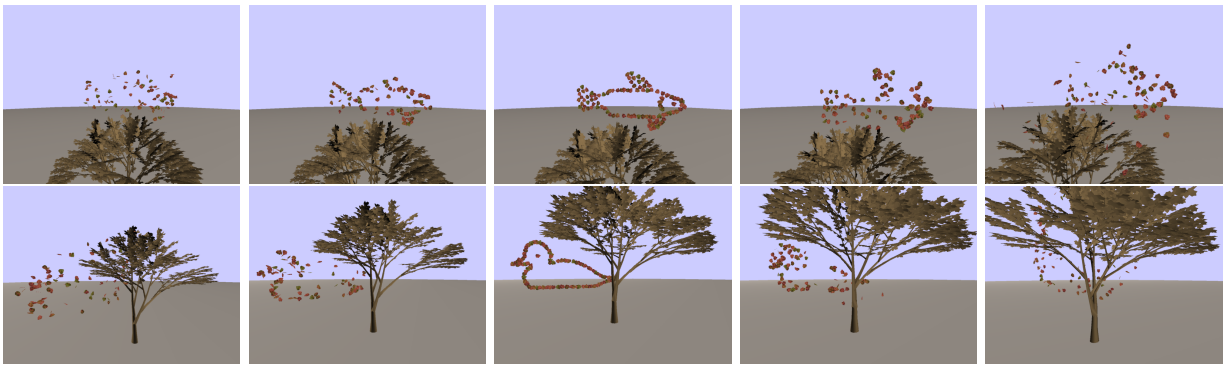


Figure 5.7: Leaves form different shapes in the air.

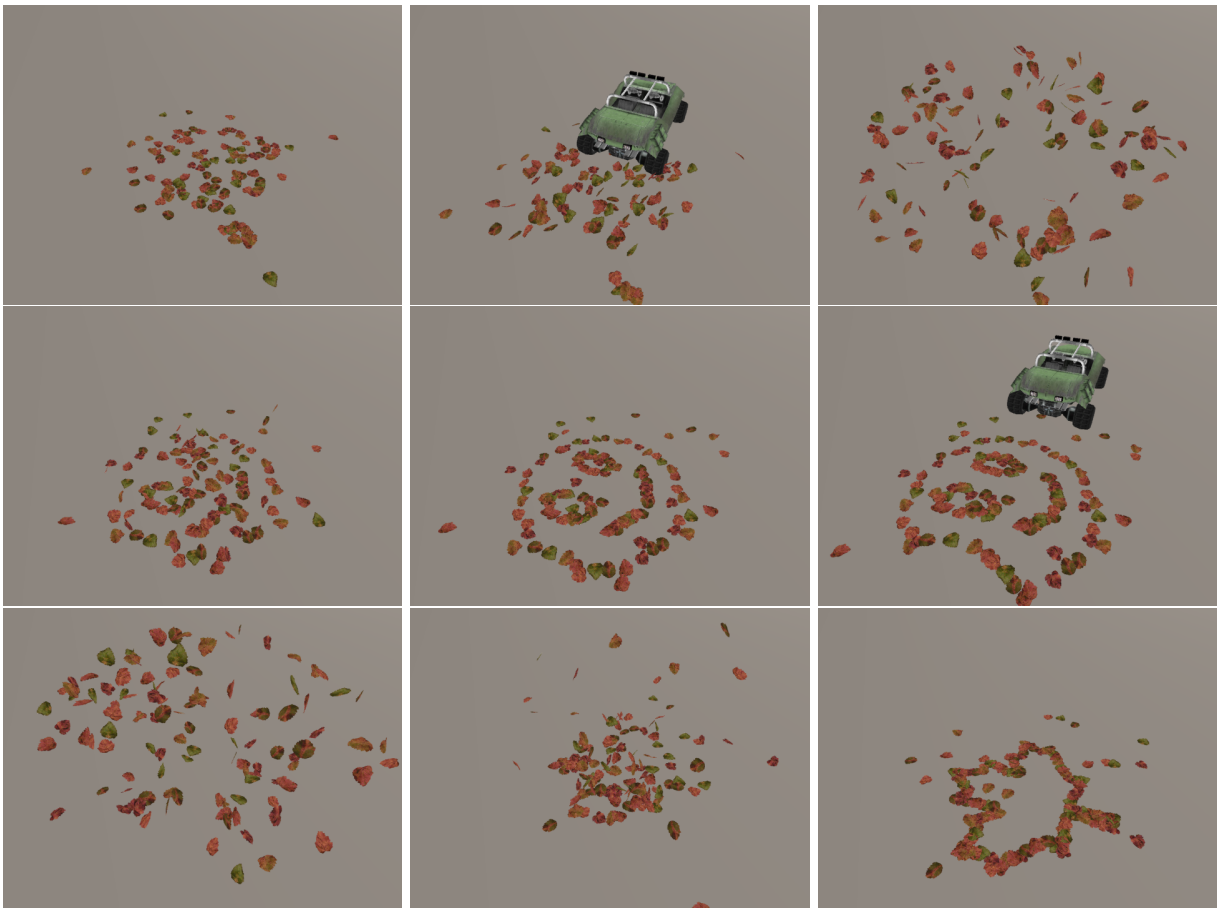


Figure 5.8: Leaves form shapes on the ground after being disturbed by external forces.

5.7 Results

To create the models used in the animations, digital photographs of leaves and feathers are used. The shapes are outlined and then triangulated, and these resulting meshes are texture-mapped with the original images. The vertices of the models are displaced in order to curve the leaves and feathers. The models are aligned with one particular axis and centered, such that any rotation is about this center of mass.

Multiple examples are shown in Fig. 5.4-5.9. These examples either involve a single object or multiple objects. Fig. 5.4 and 5.5 use one single feather to demonstrate our stylistic path planning technique. In such an example, we always specify a source, where the feather should take off, and a destination, where it is supposed to land. In addition, constraints or obstacles are also placed in the middle of the trajectory. Fig. 5.4 shows two examples with intermediate position constraints and another two examples with partially constrained trajectories. In these examples, the whole trajectory is actually divided into a few non-overlapping portions each of which has its own starting and ending points, and each non-constrained

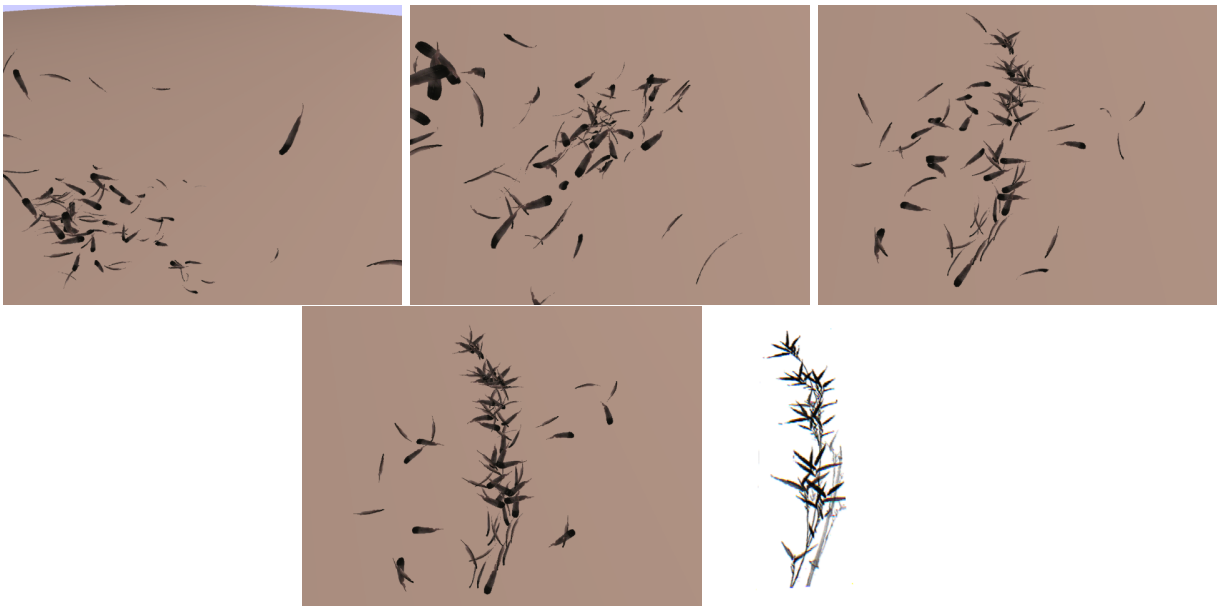


Figure 5.9: Feathers fall onto the ground, forming a pattern similar to a painting (rightmost).

portion of the trajectory is planned separately using our algorithm while a desired level of continuity between consecutive portions is maintained. Fig. 5.5 shows obstacle avoidance by placing a maze between the source and destination. Note that we still perform optimization on the planned trajectories along with additional fluid particles even though there is only one trajectory in each example. For a 300 frame trajectory, the path planning stage took less than 15 minutes on an AMD 2100+ processor. The optimization stage took approximately 2 hours.

Fig. 5.6-5.9 demonstrate fluid-like group behaviors by using multiple objects such as leaves and feathers. Typically, there is one or more keyframes in each of these examples. Each keyframe specifies the positions of all the objects. The velocities at a keyframe can also be constrained as an option. The objects have to satisfy these constraints simultaneously. The examples show two types of keyframes. The first type specifies a static configuration that should be satisfied when the objects fall to the ground while the second type is inserted in the middle of an animation when the objects are still moving. Because multiple objects indirectly interact with each other, the optimization stage becomes slower to converge and therefore, more expensive. For a 300 frame animation, it took approximately 5 hours for the optimization to converge while the motion planning stage typically took less than 2 hours.

Note that the motion database is precomputed and the same database is used for all the examples. In all the examples, the total number of gas particles is 2500 and the number of object particles is 125. The actual number of variables in the optimization is approximately six times as many as the number of particles since each particle needs six variables to describe its position and velocity. A small number of extra variables are

necessary for the parameters in the vortex model.

After optimization, the strength of the vortexes and the strength of the wind fields are typically comparable with the same order of magnitude. The wind fields change smoothly with time and location. The average duration of a vortex is about 7 seconds. The average diameter of a vortex is around one third of the width of the simulation volume.

5.8 Conclusions and Discussions

In this work, we developed a method to realistically synthesize controllable motion for a single or a group of lightweight rigid objects in a gaseous medium. We adopted an interesting formulation that considers this problem as a large-scale space-time optimization with user requirements and fluid motion equations as constraints. To make this problem tractable, we designed a novel technique to solve the optimization by taking results from a data-driven motion synthesis approach as initial solutions, and extracting vortex fields in a separate stage using both annotations and parametric model fitting. Experiments indicate our method can effectively generate desirable results.

In our experiments the user can usually obtain a satisfactory profile of the trajectories from the motion planning stage. The motion synthesis algorithm is fast (one set of results typically takes about 10 minutes) so animators can redo the motion planning and/or impose more requirements upon the system if not satisfied with the results. Once acceptable initial trajectories are obtained, optimization makes them more coherent and fluid-like. At this stage, usually only relatively small changes will be made to the motion.

Performance of the optimization system is not easy to predict given a set of trajectories and constraints. However, the cost function never increases on an iteration, so the "goodness" of the result is always improving. Animators can stop once acceptable results are generated.

We use incompressible SPH formulas to simulate air, because only the objects inside are of interest, not the whole particle set. But this fluid model is only enforced through soft constraints, so a little compressibility is allowed to make the system less stiff.

Chapter 6

Inviscid And Incompressible Fluid Simulation on Triangle Meshes

Simulating fluid motion on manifold surfaces is an interesting area with many applications. There has been previous work in the physical sciences which deals with fluid flow on a sphere, which can serve as a model for our atmosphere. In graphics, such simulations provide a means to confine fluid flows to a geometric shape and can be used for creating interesting special visual effects, such as objects with elegant dynamic appearances and soap bubbles with swirling diffraction patterns. But it is rarely explored because of the difficulty of establishing plausible physical models.

Our method simulates inviscid and incompressible fluids on meshes that are manifold surfaces. We adapt the steps in Section 2.2 from regular 3D grids to meshes with irregular connectivity. Before revising those steps, we need to clarify the discretization scheme on meshes first. Following the definition of discrete fields on meshes, we define a velocity field as a piecewise constant vector field and a pressure or density field as a piecewise linear scalar field. The constant velocity vector within each triangle is defined at the center of the triangle while pressure or density values are defined at the mesh vertices. The pressure or density at a point inside a triangle is linearly interpolated from the values at the three vertices of the triangle using the barycentric coordinates of the point as interpolation coefficients.

With such a discretization scheme on meshes, we summarize the revisions of the numerical simulation as follows.

- In the first step, we add the acceleration onto the velocity field and then solve velocity advection by implementing semi-Lagrangian tracing on the mesh surface. Details will be introduced in the next section.

- In the second step, to solve the Poisson equation for pressure on meshes, we replace the original equation with the discrete Poisson equation in (6.5). In the current context, the unknowns in the equation should be pressure values at the mesh vertices. To be consistent, we also multiply the right hand side of (6.5) with $\frac{1}{\triangle t}$. The solution to this equation is a piecewise linear pressure function defined over the mesh surface.

- In the third step, the original scheme can be followed without any change since there is a constant intermediate velocity vector within each triangle and the gradient of the pressure function is also a constant vector within each triangle. This step enforces incompressibility.
- In the last step, we also need to adapt semi-Lagrangian tracing for density advection, which will also be discussed in the next section.

These revisions produce a working numerical solution for visual fluid simulation on meshes. We will discuss in detail how to perform the advection of velocity and density fields in the next section.

6.1 Background

6.1.1 The Poisson Equation

Originally emerging from Isaac Newton's law of gravitation, the Poisson equation with Dirichlet boundary condition is formulated as

$$\nabla^2 f = \nabla \cdot \mathbf{w}, \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{6.1}$$

where f is an unknown scalar function, $\mathbf{w}$ is a *guidance* vector field, f^* provides the desirable values on the boundary of Ω, $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ is the Laplacian operator, $\nabla \cdot \mathbf{w} = \frac{\partial w_x}{\partial x} + \frac{\partial w_y}{\partial y} + \frac{\partial w_z}{\partial z}$ is the divergence of $\mathbf{w} = (w_x, w_y, w_z)$. Since the Poisson equation defines the Laplacian of an unknown *scalar* function, solving the equation is actually the process of reconstructing the scalar function from its local differential property.

6.1.2 Vector Field Decomposition

Fluid simulation is closely related to Helmholtz-Hodge vector field decomposition [70] which uniquely exists for a smooth 3D vector field $\mathbf{w}$ defined in a region Ω:

$$\mathbf{w} = \nabla\phi + \nabla \times \upsilon + \mathbf{h}, \tag{6.2}$$

where ϕ is a scalar potential field with $\nabla \times (\nabla\phi) = 0$, υ is a vector potential field with $\nabla \cdot (\nabla \times \upsilon) = \mathbf{0}$, and $\mathbf{h}$ is a field that is both divergence and curl free. The uniqueness of this decomposition requires proper boundary conditions. The scalar potential field ϕ from this decomposition happens to be the solution of the following least-squares minimization

$$\min_{\phi} \int\int_{\Omega} \|\nabla\phi - \mathbf{w}\|^2 dA, \tag{6.3}$$

whose solution can also be obtained by solving the Poisson equation, $\nabla^2\phi = \nabla \cdot \mathbf{w}$.

6.1.3 Discrete Fields and Decomposition

One prerequisite of solving differential equations over a triangle mesh is to overcome its irregular connectivity in comparison to a regular image or voxel grid. One recent approach to circumvent this difficulty is to approximate smooth fields with discrete fields first and then redefine the differential properties and equations for the discrete fields [71, 72]. A discrete vector field on a triangle mesh is defined to be piecewise constant with a constant vector within each triangle. A discrete potential field is defined to be a piecewise linear function, $\phi(\mathbf{x}) = \sum_i B_i(\mathbf{x})\phi_i$, with B_i being the piecewise-linear basis function valued 1 at vertex $\mathbf{v_i}$ and 0 at all other vertices, and ϕ_i being the value of ϕ at $\mathbf{v_i}$.

For a discrete vector field $\mathbf{w}$ on a mesh, its divergence at vertex $\mathbf{v_i}$ can be defined to be

$$(\mathrm{Div}\mathbf{w})(\mathbf{v_i}) = \sum_{T_k \in N(i)} \nabla B_{ik} \cdot \mathbf{w} |T_k| \tag{6.4}$$

where $N(i)$ is the set of triangles sharing the vertex $\mathbf{v_i}$, $|T_k|$ is the area of triangle T_k, and ∇B_{ik} is the gradient vector of B_i within T_k. Note that this divergence is dependent on the geometry and 1-ring structures of the underlying mesh.

Given the definitions of discrete fields and their divergence, the discrete Poisson equation [72] can be expressed as

$$\mathrm{Div}(\nabla\phi) = \mathrm{Div}\mathbf{w}, \tag{6.5}$$

which is actually a sparse linear system,

$$\mathbf{Af} = \mathbf{b}, \tag{6.6}$$

that can be solved numerically using the conjugate gradient method. We still call (6.5) the Poisson equation for convenience.

6.2 Advection on Meshes

Semi-Lagrangian tracing is the key component of the method we use to perform the advection of both velocity fields and density fields. In a 3D space, if we would like to obtain the advected velocity at a point $\mathbf{r}$ with the reversed velocity direction $\mathbf{u^r}$, semi-Lagrangian tracing is simply carried out by transporting the velocity at $\mathbf{r} + \mathbf{u^r}dt$ to $\mathbf{r}$, where dt is the time step.

Performing semi-Lagrangian tracing on meshes is more complicated. Since the velocity of a triangle is defined at its center, velocity tracing always starts from the center of a triangle. On the other hand, since

density (scalar) values are defined on vertices, we have to start tracing from there. However, the reversed velocity direction at a vertex is not well defined without a careful interpolation scheme. In this section, we first describe a method to advect velocities, and then generalize it to solve the advection of density (scalar) fields.

6.2.1 Advection of Velocity Fields

In a 3D space, the destination of semi-Lagrangian tracing is still in the same space. However, on a curved mesh surface, $\mathbf{r} + \mathbf{u}^{\mathbf{r}}dt$ is not necessarily on the same surface any more. To overcome this difficulty, the key point is to generate a curved trajectory on the mesh by wrapping the tracing direction around the surface and treating the shared edge of two adjacent faces properly when the trajectory crosses it from one of the faces. It is natural to require that the trajectory continue in the adjacent face from its intersection with the shared edge.

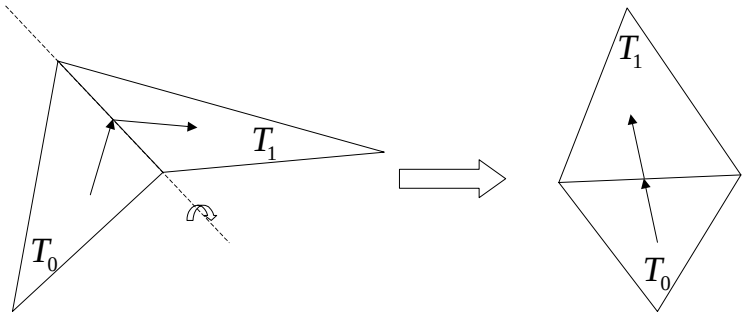


Figure 6.1: Flattening faces T_0 and T_1: rotate face T_0 to the same plane as T_1, and rotate the velocity vector using the same matrix operation.

How can we define the tracing direction on the adjacent face to guarantee continuity? The situation is shown in Fig. 6.1. We first need to define the continuity of vector fields across faces. The vector field on two adjacent faces is considered to be continuous at their shared edge if the resulting vector field from the following operation is continuous at the shared edge: the operation simply rotate one of the faces, as well as the vectors on it, along the shared edge onto the plane where the other face resides. According to this definition, it is straightforward to define the tracing direction on the adjacent face. Note that there are two possible choices in the above rotation. More reasonably we use the one that makes two faces end up on different sides of the edge. This guarantees that our algorithm degenerates into the original fluid algorithm for 2D spaces when the mesh is actually representing a flat surface. And this is an important issue when adapting the original algorithm for meshes. We call it degeneracy concern.

Once the end of the trajectory has been reached, the velocity there needs to be transported to the point

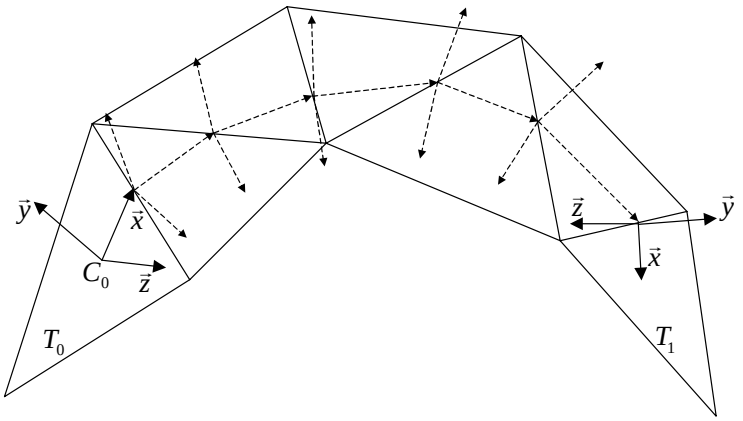


Figure 6.2: The velocity vector at the end of a traced trajectory in face T_1 is transported to the beginning of the trajectory at the center of face T_0. The x-axis of the local frame in T_0 is the starting tracing direction.

where we started tracing. In a 3D space, velocity vectors can be simply translated. However, on a mesh surface, a translated vector will not necessarily belong to the same surface any more. In addition, a curved surface change its orientation constantly. Translated vectors cannot reflect this orientation change. Our solution to this problem is based on local frames defined in the starting face and the terminating face of the trajectory. It is illustrated in Fig. 6.2. We define a local frame inside a triangle face with respect to a tracing direction as follows. The unit vector along the specific tracing direction in the face is the x-axis. The normal vector of the face is the y-axis. The z-axis is simply the cross product between the x- and y-axes. A mapping between vectors in the two faces can be defined. A vector in the second local frame is mapped to the vector in the first frame with the same local coordinates. We use this mapping to transport desirable velocity vectors from the terminating face of the trajectory to the starting face. Note that both the local frames and mapping change when the tracing direction changes. This scheme also degenerates into vector translation in 2D when the mesh is actually representing a flat surface.

Velocity Interpolation

Note here, a velocity field is represented as a piecewise constant vector field. The velocities at triangle edges are not continuous. Therefore, the velocity at the end of the traced trajectory should be first smoothly interpolated before being transported to avoid visual discontinuities and other artifacts. Before introducing our general interpolation scheme, let us first look at how to interpolate the velocity at a vertex.

The first choice we tried is to perform vector interpolation in the 3D space. Unfortunately, we found it to be problematic. Consider the situation in Fig. 6.3, where we would like to interpolate the velocity at $\mathbf{v}_4$. The result from simple 3D interpolation should be close to zero. But this conflicts with the fact that

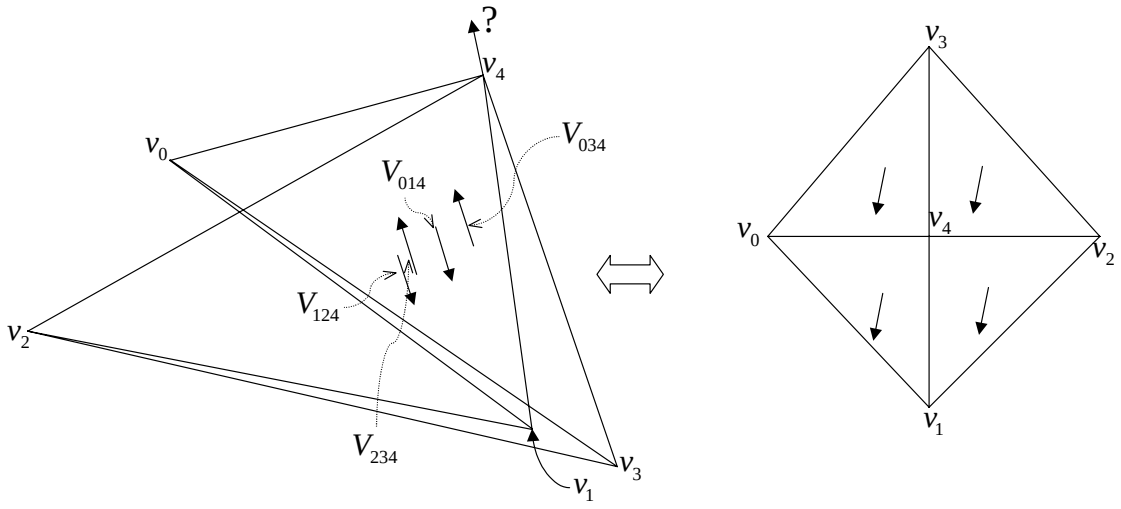


Figure 6.3: Velocity interpolation at a vertex whose neighboring faces are formed by folding a flat surface.

neighboring faces of $\mathbf{v}_4$ form a flat surface when being unfolded and in that plane all velocities are the same. Naturally, the velocity at $\mathbf{v}_4$ is expected to be the same, too.

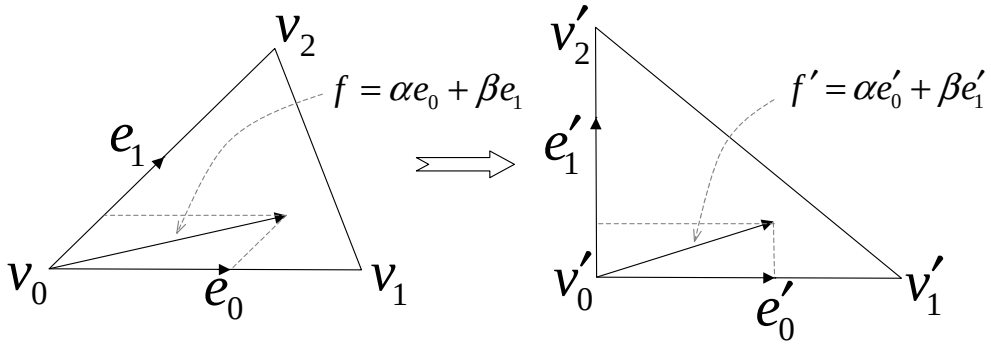


Figure 6.4: A local affine transformation defined by mapping two basis vectors $\mathbf{e}_0$ and $\mathbf{e}_1$.

To correctly interpolate the velocity at a vertex, what we really need is to flatten its 1-ring neighborhood first. Here, local flattening is achieved with affine mappings (parametrizations). In Fig. 6.4, we show how vector $\mathbf{f}$ is mapped when the triangle it lies on is transformed to a different shape. Suppose unit vectors $\mathbf{e_0}$ and $\mathbf{e_1}$ collinear with two edges of the triangle shown in the figure are transformed to $\mathbf{e'_0}$ and $\mathbf{e'_1}$. If we consider $\mathbf{e_0}$ and $\mathbf{e_1}$ as the basis vectors of an affine frame, the transformed vectors $\mathbf{e'_0}$ and $\mathbf{e'_1}$ define a new affine frame. An affine transform between the original and new affine frames can be easily derived. This affine transform can be applied to any vector defined in the original frame to obtain its corresponding new vector in the new frame. Meanwhile, the inverse affine transform is also well-defined so a vector can be mapped from the new frame back to the original frame.

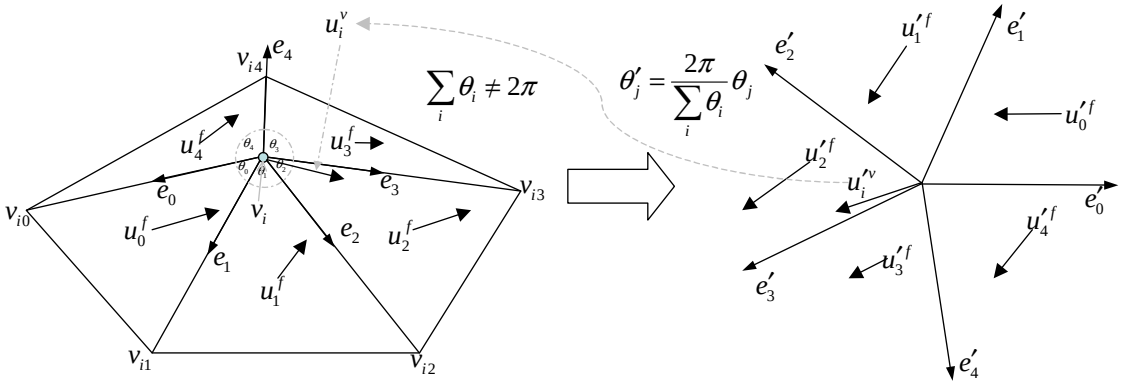


Figure 6.5: Vertex velocity interpolation in a local parametrization plane.

With these local affine mappings, velocity interpolation at a vertex proceeds as follows. We perform the unfolding process as in Fig. 6.5. We map the unit vectors pointing from $\mathbf{v}_i$ towards its neighboring vertices onto a unit circle lying on a parametrization plane such that the relative proportion of their angles is preserved. A local affine transform can be derived for each face adjacent to $\mathbf{v}_i$ using these mapped unit vectors. Then the velocity vector at each face center is mapped to a new vector on the parametrization plane using the derived affine transform for the face. The weighted average of the transformed velocity vectors is the average velocity at vertex $\mathbf{v}_i$ in the parametrization plane. The angle of a face at this vertex is used as the weight. Note that there is a distinct affine transform for each face, the average velocity in the parametrization plane cannot be uniquely mapped back onto the mesh.

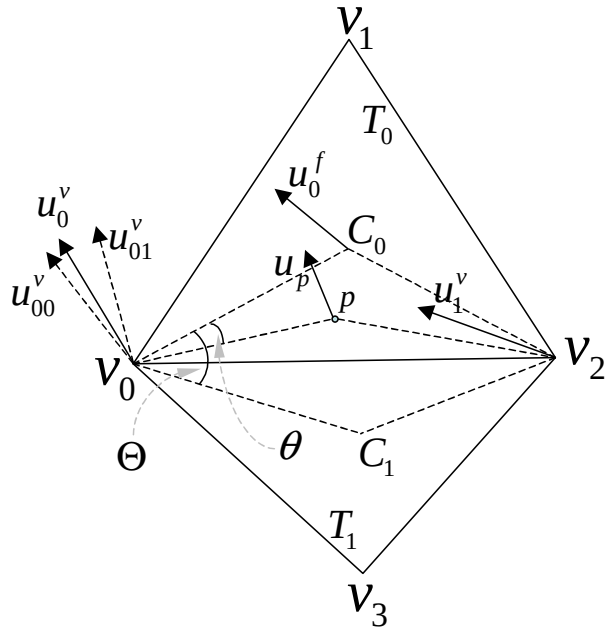


Figure 6.6: Velocity interpolation within a triangle.

Our general velocity interpolation scheme is illustrated in Fig. 6.6. Assume we are interpolating the velocity at point $\mathbf{p}$ on face T_0 whose vertices are $\mathbf{v}_0$, $\mathbf{v}_1$, and $\mathbf{v}_2$. Also assume $\mathbf{p}$ is within the triangle formed by $\mathbf{v}_0$, $\mathbf{v}_2$ and $\mathbf{c}_0$ which is the center of face T_0. We use the interpolated velocity of $\mathbf{v}_0$, $\mathbf{v}_1$ (mapped back onto face T_0) and the face velocity of T_0 at its center to interpolate that of $\mathbf{p}$. Naturally, the barycentric coordinates of $\mathbf{p}$ with respect to this smaller triangle are used as the averaging weights. To ensure the continuity of interpolated velocity across the shared edge of two adjacent faces, this scheme needs to be enhanced.

Suppose T_0 and T_1 are two adjacent faces, and $\mathbf{v}_0$ and $\mathbf{v}_2$ are the vertices of their shared edge. If we flatten the 1-ring structure at $\mathbf{v}_0$, there are two distinct affine mappings associated with T_0 and T_1, respectively. Vertex $\mathbf{v}_0$ has an average velocity in its local parametrization plane. The inverse affine transform associated with T_0 maps this average velocity back to a vector $\mathbf{u}_{00}^v$ coplanar with T_0. A vector $\mathbf{u}_{01}^v$ coplanar with T_1 can also be generated similarly. In general, $\mathbf{u}_{00}^v$ and $\mathbf{u}_{01}^v$ are not the same even if we flatten T_0 and T_1 into a plane along their shared edge. We decided to linearly interpolate between these two vectors to achieve continuous interpolated velocity at the shared edge. Suppose the center of T_0 is $\mathbf{c}_0$ and the center of T_1 is $\mathbf{c}_1$. Let θ be the angle between lines $\mathbf{v_0p}$ and $\mathbf{v_0c_0}$, Θ be the angle between lines $\mathbf{v_0c_1}$ and $\mathbf{v_0c_0}$ after flattening. The interpolation coefficient between $\mathbf{u}_{00}^v$ and $\mathbf{u}_{01}^v$ is defined to be $\frac{\theta}{\Theta}$. Thus, the interpolated velocity at $\mathbf{v}_0$ is

$$\mathbf{u}_0^v = \left(1 - \frac{\theta}{\Theta}\right)\mathbf{u}_{00}^v + \frac{\theta}{\Theta}\mathbf{u}_{01}^v. \tag{6.7}$$

The interpolated velocity $\mathbf{u}_2^v$ at $\mathbf{v}_2$ is defined similarly. The final interpolated velocity $\mathbf{u_p}$ at $\mathbf{p}$ is

$$\mathbf{u_p} = \alpha\mathbf{u}_0^v + \beta\mathbf{u}_2^v + (1 - \alpha - \beta)\mathbf{u}_0^f \tag{6.8}$$

where $\mathbf{u}_0^f$ is the face velocity at the center of T_0, and $(\alpha, \beta, 1 - \alpha - \beta)$ represent the barycentric coordinates of $\mathbf{p}$ inside the triangle formed by $\mathbf{v}_0$, $\mathbf{v}_1$ and $\mathbf{c}_0$.

6.2.2 Advection of Scalar Fields

At the last substep of each time step, the scalar density field is advected and interpolated. This part only differs from the advection of the velocity field in two aspects. For a density field defined on vertices, it is not obvious from which face to start the tracing and with what velocity. We use the average vertex velocity in its parametrization plane to determine the face where the velocity vector lies and map the average velocity back onto it. And the rest of tracing is exactly the same as that of velocity field. Unlike velocity advection, we do not need to transport vectors and perform vector interpolation. Density interpolation at the end of

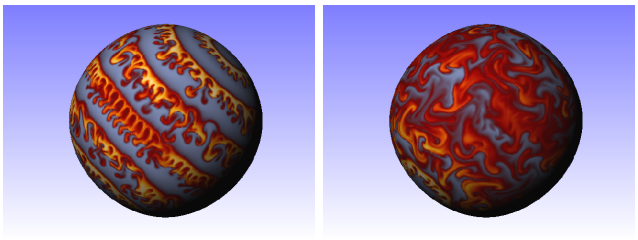


Figure 6.7: Fluid flows on a Sphere model with a striped initial density distribution. The mesh has 98306 vertices and 196608 faces.

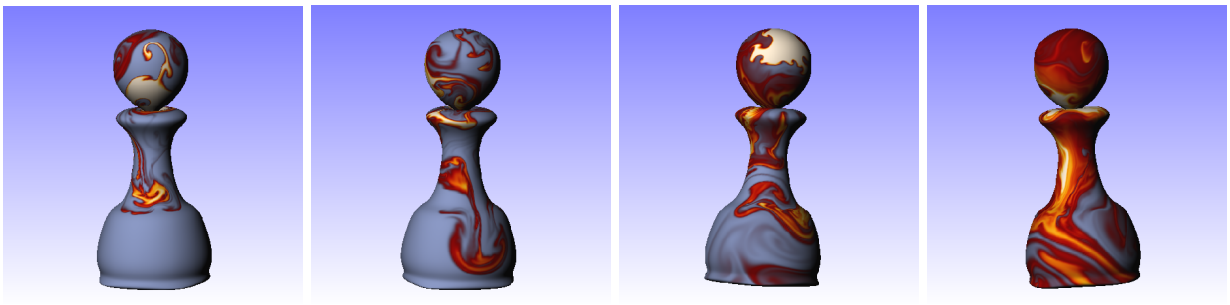


Figure 6.8: Density is periodically deposited onto the top of a Pawn model, and gravity drives the rest of the simulation. The mesh has 130050 vertices and 260096 faces.

the trajectory is much simpler since density values are defined on vertices instead of face centers. We only need to use the barycentric coordinates of the endpoint of the trajectory with respect to the vertices of the terminating face to interpolate from the density values at those vertices. The interpolated density is deposited onto the vertex where the tracing started.

6.3 Implementation

The sparse matrix equation in (6.6) can be solved using either conjugate gradient (CG), or preconditioned conjugate gradient (PCG). In the case of PCG, we use the Incomplete Cholesky Factorization (ICF) as in [29].

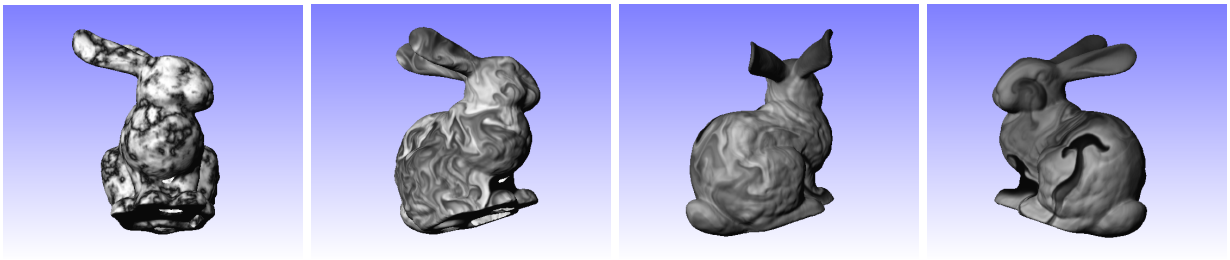


Figure 6.9: Fluid flows on a Bunny model. The mesh has 139122 vertices and 277804 faces.

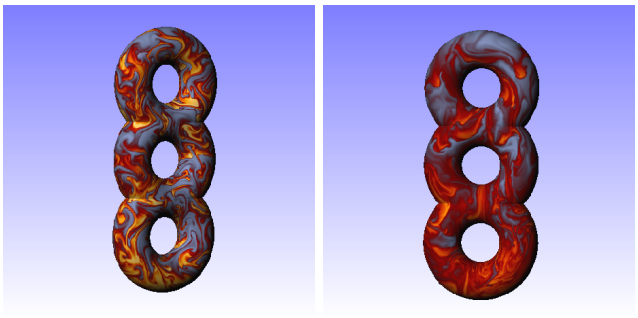


Figure 6.10: Fluid flows on a high genus Torus model. The mesh has 271356 vertices and 542720 faces.

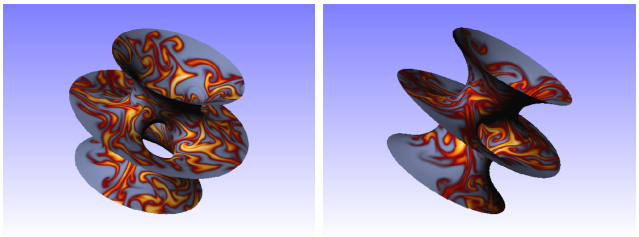


Figure 6.11: Fluid flows on the Hypersheet model. The mesh has 117877 vertices and 234752 faces.

To handle meshes with open boundaries, a few places in the implementation need to be modified. When solving the Poisson equation for pressure, the pressure values on the mesh boundaries are actually the boundary condition for the Poisson equation. In practice, we use Dirichlet boundary conditions and set the pressure to be 0 at boundary vertices. The coefficient matrix in (6.6) needs to be adjusted to accommodate this type of boundary conditions. When setting up the local affine transforms during velocity interpolation at boundary vertices, we keep the original angles when they are mapped into the parametrization plane unless the summation of all the angles surrounding a vertex exceeds 2π, which in practice we did not notice any. Semi-Lagrangian tracing can go beyond the mesh boundary, in which case smoothly decreasing the density helps prevent undesirable discontinuities. Note that our current implementation does not enforce zero divergence at the boundary vertices.

6.4 Results

Listed here are the results of several examples. In the Pawn example shown in Fig. 6.8, we periodically deposit density onto the top of it, and let gravity drive the rest of the simulation. The model is derived from an original Pawn model after five iterations of Loop subdivision. If we run our program on the original coarse Pawn model with 16256 faces, we can achieve a frame rate of 6fps on a 1.7GHz Pentium IV Xeon processor. Because of incompressibility, the fluid moves in the opposite direction of gravity in some regions.

In terms of the bunny model shown in Fig. 6.9, we initially place density as horizontal strips, and use gravity to pull them down. Note that the bunny model is not a waterproof model and has some holes on the bottom and between the forelegs. The fluid moves interestingly around these regions. The fluid behavior at open mesh boundaries is even more clear in the hypersheet model shown in Fig. 6.11 where the fluid escapes when crossing the boundary. We use Dirichlet boundary condition for both examples. Note that it is also a straightforward implementation to reflect the fluid velocity vector once it reaches the boundary. In the triple torus example shown in Fig. 6.10, we show our method works equally well for high genus models.

6.5 Discussions

We described a novel fluid simulation method directly performed on triangle meshes. It eliminates parametrization distortions and enforces incompressibility on closed surfaces. This work can be potentially generalized to other types of manifold meshes and even unstructured 3D grids. So far our algorithm works for inviscid and incompressible fluid simulation. For a more complicated and realistic fluid solver, further studies of the underlying physics foundation is essential. We would also like to improve the performance of our solver especially by using an appropriate preconditioner for the conjugate gradient method.

Chapter 7

A Fast Multigrid Algorithm for Mesh Deformation

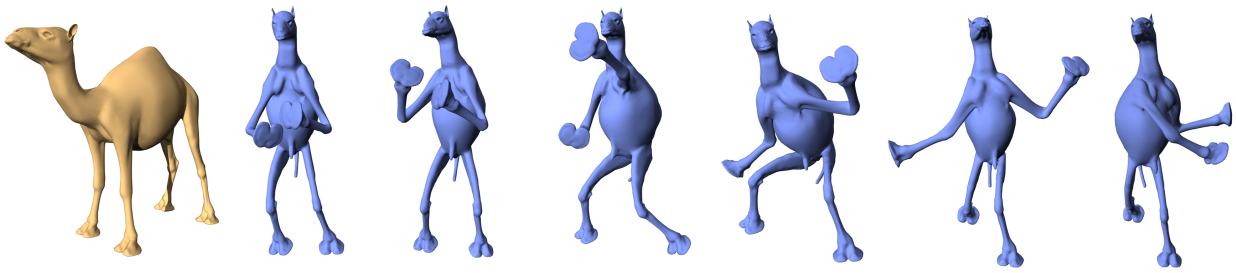


Figure 7.1: The idle CAMEL becomes a boxer with the help of MOCAP data and our mesh deformation system.

In this chapter, we present a multigrid technique for efficiently deforming large surface and volume meshes. We show that a previous least-squares formulation for distortion minimization reduces to a Laplacian system on a general graph structure for which we derive an analytic expression. We then describe an efficient multigrid algorithm for solving the relevant equations. Here we develop novel prolongation and restriction operators used in the multigrid cycles. Combined with a simple but effective graph coarsening strategy, our algorithm can outperform other multigrid solvers and the factorization stage of direct solvers in both time and memory costs for large meshes. It is demonstrated that our solver can trade off accuracy for speed to achieve greater interactivity, which is attractive for manipulating large meshes. Our multigrid solver is particularly well suited for a mesh editing environment which does not permit extensive precomputation. Experimental evidence of these advantages is provided on a number of meshes with a wide range of size. With our mesh deformation solver, we also successfully demonstrate that visually appealing mesh animations can be generated from both motion capture data and a single base mesh even when they are inconsistent.

7.1 Introduction

Surface-based mesh editing has received much attention recently due to its capability to produce visually appealing results while at the same time making the underlying numerical computation transparent to the

user. The user only needs to specify the goals, which can be handle positions, key frames or silhouettes, and the editing system automatically solves a sparse system of equations to satisfy these constraints. Nevertheless, a serious problem that still hampers the deployment of this type of techniques is their scalability. When meshes become large and complex, the performance of the numerical solver becomes the bottleneck of the entire system. While solutions to cope with this problem exist – including small regions of interest (ROI) and precomputed matrix factorizations – they restrict the scope of editing operations.

The multigrid method on the other hand has the potential to solve large-scale sparse systems efficiently without a significant setup time. While the user most often only needs to pick up a handle to manipulate a mesh, it is sometimes necessary to define and manipulate new handles that have not been preprocessed. Furthermore, in a general mesh editing environment, mesh deformation needs to be mixed with other mesh editing operations, such as remeshing and merging. This is evidenced by routine practice in game development where large-scale meshes are edited first before simplified to a size suitable for real-time rendering. Some of these editing operations result in an altered system of linear equations that need to be solved on the fly. When the mesh is large, a fast multigrid algorithm can solve the altered linear system in a less stressful way than the factorization stage in direct solvers.

In this chapter, we introduce a fast multigrid technique tailored for mesh deformation to support the aforementioned scenario. Although the multigrid method has become a popular choice for large-scale mesh processing [24, 25], there are still a number of challenges we need to overcome to achieve acceptable interactive performance. First, recent mesh deformation techniques often have two passes with the first pass solving for local frames and the second pass solving for vertex coordinates. How can we effectively reformulate these passes so that they become more compatible with multigrid solvers? Second, the multigrid method requires a hierarchical structure and must accommodate user-provided deformation constraints within this hierarchy. How can we properly handle these aspects? Third, when moving between grid levels the multigrid method applies a pair of prolongation and restriction operators. How should we design such operators to speed convergence?

We have developed effective techniques to overcome these challenges. First, we revise the formulations for mesh deformation so that they can be adequately solved by local relaxations. In particular, we analytically obtain a closed-form formulation for the optimization of vertex coordinates, thereby avoiding expensive sparse matrix multiplications. Second, we quickly create a hierarchy using a simple graph coarsening technique that ignores the initial mesh structure. Boundary conditions are not explicitly considered during the construction of the hierarchy. Instead, they are incorporated algebraically in the equations and in the prolongation/restriction operators. Most importantly, we develop a novel technique that automatically obtains

prolongation and restriction operators using a weighted graph perspective. These operators better maintain the consistency of equations among different levels, and thus significantly improve the convergence rate. As a result, our algorithm can outperform existing multigrid solvers and the factorization stage of direct solvers. We demonstrate the advantages and utility of these features in complex mesh editing examples.

7.1.1 Related Work

There have been many approaches for mesh modeling and editing. In the following, we will focus on geometry-based mesh deformation techniques only. Physically based deformation based on elasticity models is beyond the scope of this chapter.

FFD techniques [73] need to embed a surface mesh inside a volume lattice. It is hard to precisely control the surface deformation results using FFD because it is achieved indirectly by manipulating the lattice points. Recent work [74] has replaced the lattice with volume-based radial basis functions (RBFs) which further induce deformation on the target mesh surface. Impressive performance on large meshes has been achieved for deformations with predefined handles. Flexible mesh modeling and deformation can be achieved by employing a multiresolution decomposition of the original mesh [75, 76, 77]. By choosing to work at an appropriate resolution, one can manipulate or edit the mesh at a desired scale to reduce the amount of manual effort. Recently, surface-based mesh editing techniques [78, 79, 80, 7, 81] demonstrate that it is possible to achieve similar results without exposing the multiresolution structure of the mesh. The user directly manipulates the mesh surface at its finest level and the scale of manipulation is controlled by a region of interest. High frequency mesh details can be well preserved by locally or globally supported deformation fields which only modify the low frequency part of the mesh geometry.

At the core of a surface-based mesh editing system lies a mesh representation based on differential coordinates, such as Laplacian or gradient coordinates. Such systems typically manipulate the differential coordinates of the undeformed mesh first, followed by a reconstruction of the deformed mesh from the modified differential coordinates. Most techniques in this category have two passes with the first pass addressing rotation and scaling, and the second pass reconstructing vertex coordinates. While the second pass is commonly formulated as a least-squares minimization, various techniques have been proposed for the first pass, including geodesic distance based propagation [80], harmonic field based interpolation [81], and least-squares minimization based on a rotation-invariant representation [7]. In [82], a volume graph is constructed at the interior of a closed mesh to prevent volume loss during excessive bending and twisting.

Mesh deformation is also closely related to shape interpolation which involves more than one originally undeformed key shapes. An effective shape interpolation technique for simplicial complexes was introduced

in [83]. This technique has been generalized to surface-based deformation transfer in [57] and nonlinear interpolation among multiple key shapes in [84].

7.2 Basic Formulations

Here we describe the basic formulations we adopt for mesh deformation. Note that these formulations can be applied to surface triangle meshes with or without a volume graph [82]. In practice, we most often construct such a graph to have better volume preservation. Following [7], an initial local frame is defined at each vertex of the graph. The orientation of the local frame can be arbitrary. Given a set of rotation, scaling and/or translation constraints, we still formulate mesh deformation as a two-pass process. During the first pass, we first compute harmonic guidance fields over the mesh as in [81], and then obtain modified orientation and scale of the local frame at every unconstrained vertex by interpolating relevant constraints using the harmonic guidance fields. The modified orientation and scale are then fixed at all vertices during the second pass and the coordinates of every unconstrained vertex are solved as in [7]. More details follow.

7.2.1 The First Pass

During the first pass, the goal is to smoothly "interpolate" both rotation and scaling constraints over the entire mesh surface. We perform interpolation for these two types of constraints separately because the orthogonality of local frames requires that rotation interpolation still produce valid rigid body rotations. The interpolation of scaling constraints is performed by computing a single scalar harmonic field, which is the solution of the Laplace equation,

$$\Delta\nu = 0, \tag{7.1}$$

over the mesh surface with the scaling constraints as boundary conditions. The discretization of this Laplace equation gives rise to a sparse linear system.

As we know, a harmonic field is the equilibrium state of a diffusion process. Ideally, one should generalize this observation from scaling constraints to rotation constraints each of which can be represented as a unit quaternion. We define an orientation diffusion process as follows,

$$\frac{\partial \mathbf{q}}{\partial t} = \Delta_O \mathbf{q}, \tag{7.2}$$

where Δ_O represents a generalized Laplacian operator for orientations. Since a discrete Laplacian operator at a mesh vertex returns a linear combination of the values of a quantity within the 1-ring neighborhood of the

vertex, such a Laplacian operator can be easily generalized to orientations because any linear combination of unit quaternions is well defined as long as the participating quaternions follow a predefined order and the weights form a partition of unity. Iteratively simulating such an orientation diffusion process as defined in (7.2) with appropriate boundary conditions leads to a smooth quaternion field over the mesh. Each iteration of the simulation is a local relaxation. The resulting quaternion field can then be applied to the local frame at every vertex to obtain the modified local frame.

However, orientation diffusion using quaternions is still more expensive than scalar diffusion. In practice, we adopt one of the schemes presented in [81] which computes a distinct bounded scalar harmonic field for each and every rotation constraint. Each of the bounded harmonic fields is computed using a unique set of boundary conditions which set one at the vertices sharing the same rotation constraint being considered while zero at the vertices with the rest of the rotation constraints. The value of the resulting harmonic field at an unconstrained vertex serves as the interpolation coefficient for the rotation constraint being considered. The final rotation at that vertex is interpolated from all rotation constraints using the computed coefficients. Such an interpolated rotation is a good approximation of the rotation obtained from orientation diffusion.

7.2.2 The Second Pass

Our formulation for solving modified vertex coordinates in the deformed mesh is based on the method in [7]. Suppose vertex $\mathbf{v}_j$ belongs to vertex $\mathbf{v}_i$'s 1-ring neighborhood. The coordinates of $\mathbf{v}_i$ and $\mathbf{v}_j$ are denoted as $\mathbf{x}_i$ and $\mathbf{x}_j$. The modified local frame at $\mathbf{v}_j$ has three axes, $\mathbf{b}_1^j$, $\mathbf{b}_2^j$ and $\mathbf{N}^j$. A linear equation that specifies the desired relative position between $\mathbf{v}_i$ and $\mathbf{v}_j$ in the deformed mesh can be formulated as

$$\mathbf{x}_i - \mathbf{x}_j = c_{ji}^1 \mathbf{b}_1^j + c_{ji}^2 \mathbf{b}_2^j + c_{ji}^3 \mathbf{N}^j, \tag{7.3}$$

where c_{ji}^1, c_{ji}^2 and c_{ji}^3 are scalar coefficients encoding the relative position of $\mathbf{v}_i$ with respect to the local frame at $\mathbf{v}_j$ in the original undeformed mesh. If we switch the roles of $\mathbf{x}_i$ and $\mathbf{x}_j$ in (7.3), we can obtain a second equation for the same vertices as follows,

$$\mathbf{x}_j - \mathbf{x}_i = c_{ij}^1 \mathbf{b}_1^i + c_{ij}^2 \mathbf{b}_2^i + c_{ij}^3 \mathbf{N}^i. \tag{7.4}$$

Subtracting (7.4) from (7.3), we have

$$\mathbf{x}_i - \mathbf{x}_j = \mathbf{d}_{ji}, \tag{7.5}$$

where

$$\mathbf{d}_{ji} = \frac{1}{2}\left(\left(c_{ji}^1\mathbf{b}_1^j + c_{ji}^2\mathbf{b}_2^j + c_{ji}^3\mathbf{N}^j\right) - \left(c_{ij}^1\mathbf{b}_1^i + c_{ij}^2\mathbf{b}_2^i + c_{ij}^3\mathbf{N}^i\right)\right). \tag{7.6}$$

Since there is such a linear equation for every pair of connected vertices, a least-squares solution can be sought for all the vertex coordinates in the deformed mesh. Using a matrix multiplication to obtain the normal equations of this least-squares minimization is actually a very costly step. Fortunately, in this particular case, it is straightforward to obtain the normal equations analytically.

From a different perspective, (7.5) can be viewed as an equation that provides a prediction of $\mathbf{v}_i$'s location using the coordinates and local frame at $\mathbf{v}_j$. Suppose $\mathbf{v}_i$'s 1-ring neighbors are indexed by $N(i)$. Obviously, there are $|N(i)|$ potentially conflicting predictions like this for $\mathbf{v}_i$. Suppose the edge between $\mathbf{v}_i$ and $\mathbf{v}_j$ is associated with a weight w_{ij} and $w_{ij} = w_{ji}$. The (weighted) least-squares solution of $\mathbf{v}_i$'s coordinates should be the (weighted) average of the $|N(i)|$ predictions. Therefore, the system of normal equations must contain exactly one equation like the following for every vertex.

$$\mathbf{x}_i = \frac{\sum_{j\in N(i)} w_{ji}\mathbf{x}_j}{\sum_{j\in N(i)} w_{ji}} + \frac{\sum_{j\in N(i)} w_{ji}\mathbf{d}_{ji}}{\sum_{j\in N(i)} w_{ji}}. \tag{7.7}$$

We will solve these derived normal equations in our multigrid solver. Actually, (7.7) has already been formulated as a type of local relaxation that a multigrid solver can immediately use. Interestingly, (7.7) can also be reformulated as

$$\sum_{j\in N(i)} w_{ji}(\mathbf{x}_j - \mathbf{x}_i) = -\sum_{j\in N(i)} w_{ji}\mathbf{d}_{ji}, \tag{7.8}$$

which reveals that it is actually a discretized Poisson equation with the left hand side formulated as a discretized Laplacian operator for the deformed mesh. The right hand side informs how to adapt the original Laplacian that was obtained from the undeformed mesh. It does not simply apply the local transformation at $\mathbf{v}_i$ to the original Laplacian. The coefficient matrix of the linear system in (7.8) is symmetric and positive definite. A few weighting schemes for the Laplacian operator have been discussed in [82] and can be adopted here. We have found experimentally that even uniform weighting can give rise to successful solutions of our system without artifacts. Since this discretized Poisson equation is based on edges, it can be used for deforming any graph structures, not just triangle meshes. Therefore, it is more general than the discretized Poisson equation adopted in [80] which was tailored for triangle meshes. Our formulation is also different from the one used in Laplacian mesh editing [79] which solves a bi-harmonic system.

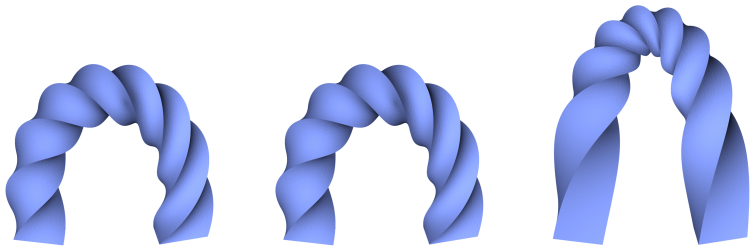


Figure 7.2: Orientation diffusion (left) and harmonic fields (center) produce more natural deformations than the method in [7]. Two constrained handles are located at the two ends of the mesh structure.

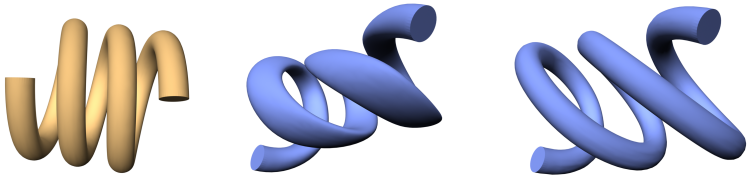


Figure 7.3: A SPRING is deformed using a spatial deformation technique (center) and our method (right). In the spatial case, points in close proximity (Euclidean metric) move together even when their geodesic distance is much larger. In contrast, our mesh-based approach contains this geodesic information implicitly.

7.2.3 Discussion

We have performed two simple tests to justify the choices we made here. Fig. 7.2 shows a comparison of three deformation results. We use the same linear system in (7.8) as the second pass in all three tests. The first pass adopts three different techniques including orientation diffusion, bounded harmonic fields [81], and the linear system in [7]. The results from the first two are extremely close and distribute both twisting and bending more uniformly across the surface than the third approach which tends to concentrate deformation halfway between the two handles. This is because the method in [7] does not enforce the orthogonality of the modified local frames. In another simple test shown in Fig. 7.3, we compare a space deformation technique [74] with the two-pass surface deformation adopted in this work. The result from the first technique produces undesirable deformation on the SPRING model because it adopts Euclidean distance which is not always a good estimate of geodesic distance. Defining a local region of interest to avoid this problem is not always feasible for models such as the one shown here.

7.3 The Multigrid Method

The multigrid method was originally introduced to solve linear systems that arise from discretizations of elliptic PDEs. As we know, both linear and nonlinear equations can be solved by iterative relaxations. Unfortunately, these iterative solvers exhibit slow convergence in large-scale problems. The multigrid method [85] can significantly improve the efficiency of these iterative solvers by accelerating the global propagation

Algorithm 1: The multigrid method.

Data: Given $\mathbf{A}_h$, $\tilde{\mathbf{u}}_h$, $\mathbf{b}_h$, ν_1, ν_2 and γ;
Result: Return $\mathbf{u}_h$ that satisfies $\mathbf{A}_h\mathbf{u}_h = \mathbf{b}_h$.

Multigrid($\mathbf{A}_h$, $\tilde{\mathbf{u}}_h$, $\mathbf{b}_h$, ν_1, ν_2, γ)
begin
  if *Coarsest Level* **then**
    return $\mathbf{u}_h$ = Solve($\mathbf{A}_h$, $\mathbf{b}_h$);
  end
  else
    for *i=1 to* γ **do**
      $\tilde{\mathbf{u}}_h$ = Smooth($\mathbf{A}_h$, $\tilde{\mathbf{u}}_h$, $\mathbf{b}_h$, ν_1);
      $\Delta\mathbf{b}_h = \mathbf{b}_h - \mathbf{A}_h\tilde{\mathbf{u}}_h$;
      $\Delta\mathbf{b}_{2h}$ = Restriction($\Delta\mathbf{b}_h$);
      Compute $\mathbf{A}_{2h}$;
      $\Delta\mathbf{u}_{2h}$ = Multigrid($\mathbf{A}_{2h}$, $\mathbf{0}$, $\Delta\mathbf{b}_{2h}$, ν_1, ν_2, γ);
      $\Delta\mathbf{u}_h$ = Prolongation($\Delta\mathbf{u}_{2h}$);
      $\tilde{\mathbf{u}}_h = \tilde{\mathbf{u}}_h + \Delta\mathbf{u}_h$;
      $\tilde{\mathbf{u}}_h$ = Smooth($\mathbf{A}_h$, $\tilde{\mathbf{u}}_h$, $\mathbf{b}_h$, ν_2);
    end
    return $\mathbf{u}_h = \tilde{\mathbf{u}}_h$;
  end
end

of information. It takes advantage of multiple discrete formulations of a numerical problem over a range of resolution levels. The coarser levels trade spatial resolution for direct communication paths over larger distances. Multigrid techniques have several important components: i) a hierarchy of discrete formulations over a range of spatial resolutions, ii) a local iterative smoothing operator, such as the Gauss-Seidel or Jacobi relaxation, iii) a prolongation operator that interpolates solutions from coarse resolutions to finer ones, and iv) a restriction operator that subsamples residual errors at finer resolutions onto coarser ones.

To be more concrete, let us consider a sparse linear system of equations, $\mathbf{A}_h\mathbf{u}_h = \mathbf{b}_h$, where $\mathbf{u}_h$ represents the vector of unknowns defined on the finest 2D rectangular grid and h represents the grid spacing there. A pyramid of grids can be defined by reducing the resolution of the finer grid by half every time. Suppose we have obtained an initial guess of the solution at the finest level, $\tilde{\mathbf{u}}_h$ (which could simply be $\mathbf{0}$). This initial solution is smoothed by one or more iterations of the local smoothing operator. Then we need to solve the defect equation, $\mathbf{A}_h\Delta\mathbf{u}_h = \Delta\mathbf{b}_h$, where the residual errors are $\Delta\mathbf{b}_h = \mathbf{b}_h - \mathbf{A}_h\tilde{\mathbf{u}}_h$, and $\Delta\mathbf{u}_h$ represents the correction to the initial guess. Suppose we have defined a pair of linear operators: a restriction operator $\mathbf{R}$ and a prolongation operator $\mathbf{P}$. Instead of solving the defect equation on the finest level, we first subsample the current residual errors onto the next coarser level using the restriction operator, $\Delta\mathbf{b}_{2h} = \mathbf{R}\Delta\mathbf{b}_h$, and

then recursively solve the following restricted equation on the coarser level,

$$\mathbf{A}_{2h}\Delta\mathbf{u}_{2h} = \Delta\mathbf{b}_{2h}, \tag{7.9}$$

where $\mathbf{A}_{2h}$ is an appropriate approximation of $\mathbf{A}_h$ on the coarser level, and is typically defined as follows,

$$\mathbf{A}_{2h} = \mathbf{R}\mathbf{A}_h\mathbf{P}. \tag{7.10}$$

Once this recursive process has reached the coarsest level, a direct solver is used to obtain an accurate solution there. These steps are summarized in Algorithm 1. Further details on multigrid solvers can be found in many excellent books and tutorials, including [85, 86].

The above procedure only illustrates the steps to solve the defect equation once an initial guess of the solution has been obtained. How can we obtain a good initial guess? Simply using the zero vector is actually not very efficient. Demonstrated in Fig. 7.4, the *full* multigrid algorithm (FMG) addresses this problem. FMG first obtains a restricted equation at each level of the hierarchy by recursively applying (7.10) to the original equation (instead of the defect equation) at the finest level. It then starts from the coarsest level and obtains an accurate solution of the restricted equation there. Once an accurate solution is obtained at a coarse level, it is interpolated onto the next finer level using the prolongation operator. This interpolated solution serves as the initial guess at that level, which then calls the multigrid method in Algorithm 1 to construct a defect equation specific to its own level and obtain an accurate solution to its own restricted equation. This process terminates once it reaches the finest level and obtains an accurate solution there. Note that the full multigrid algorithm obtains an accurate solution at each level, but does so very efficiently by invoking the recursive multigrid method instead of running iterative relaxations until convergence.

In some cases the discretization of a problem suggests a natural geometric coarsening. For instance, the uniform 2D grid can be coarsened by doubling the grid spacing at each level of the hierarchy. Unfortunately, irregular domains with complex discretizations do not admit such simple rules. *Algebraic* multigrid (AMG) methods have been developed in response to this problem. In AMG, the geometry of the problem is ignored and only the associated linear system is used to determine the multigrid hierarchy.

Multigrid techniques have been applied to graphics related problems in many occasions. One of the earliest applications of multigrid techniques to geometric modeling was developed in [76] which applies cascadic multigrid to mesh fairing. Related cascadic multigrid techniques for computing conformal maps and fair Morse functions on unstructured meshes can be found in [24, 87]. Efficient multilevel solvers for unstructured meshes has been introduced in [25] which relies on two new mesh hierarchies to achieve fast convergence. All

these geometric multigrid or multilevel techniques exploit mesh simplification steps for the construction of mesh hierarchies. An effective geometric multigrid algorithm with weighted prolongation/restriction operators has recently be developed in [88] for solving 2D inhomogeneous Laplace equation on 2D regular grids. AMG techniques for surface reconstruction or feature-based mesh decomposition can be found in [89, 90]. Multigrid solvers for regular grids have also been mapped onto GPUs [91, 92].

7.4 Graph Coarsening

Multigrid techniques require a hierarchy of progressively coarser grids. In our context, the grid at level l of the hierarchy is an unstructured graph, $G^l = (V^l, E^l)$, with a set of vertices, V^l, and edges, E^l. Note that our original input is a surface triangle mesh, potentially enhanced with a volume graph, and a set of constraints. We construct the finest grid in the hierarchy by only including the unconstrained vertices from the input as well as the edges connecting two unconstrained vertices. As a result, coarsened grids at higher levels do not directly involve constrained vertices either. Nevertheless, constraints are still precisely represented and satisfied in the equations. For example, once a subset of the original vertices are constrained to fixed positions in (7.8), the coordinates of the constrained vertices are not unknowns any more. They become part of the boundary condition, and should be moved to the right hand side of the equation. Thus, (7.8) can be further reformulated as follows if position constraints are taken into account,

$$\sum_{j \in N(i) \setminus C(i)} w_{ji}(\mathbf{x}_i - \mathbf{x}_j - \mathbf{d}_{ji}) + \alpha_i \mathbf{x}_i = \beta_i, \tag{7.11}$$

where $C(i)$ indexes the constrained vertices in $\mathbf{v}_i$'s original 1-ring neighborhood, $\mathbf{x}_i$'s are the unknowns, $\alpha_i = \sum_{j \in C(i)} w_{ji}$, and $\beta_i = \sum_{j \in C(i)} w_{ji}(\mathbf{x}_j + \mathbf{d}_{ji})$. As we can see, position constraints have been accumulated into the constants, α_i and β_i, in (7.11) and will be enforced when we solve the reformulated equation. Constraints during the first pass can be incorporated in a similar way. If constraints from the two passes share the same subset of original vertices, only one hierarchy needs to be constructed; otherwise, a distinct hierarchy is constructed for each pass.

Once we have constructed the finest grid, which is actually a graph with all the free vertices, we do not maintain the original mesh structure during coarsening. In practice, this does not produce inferior mesh deformation results. Our graph coarsening is based on maximal δ-independent vertex set. A subset of vertices, V^l_{ind}, is δ-independent if for any $\mathbf{v}^l_i, \mathbf{v}^l_j \in V^l_{ind}$, $e^l_{ij} \notin E^l$ or $\|\mathbf{v}^l_i - \mathbf{v}^l_j\| > \delta^l$. Our graph coarsening simply chooses a maximal δ-independent vertex set to be the vertex set for the graph at the next coarser level. Suppose the expected percentage of retained vertices after each level of coarsening is r. The distance

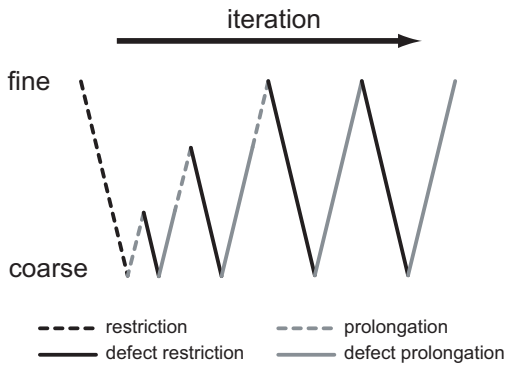


Figure 7.4: This figure illustrates the sequence of restriction and prolongation operators in a three-level, full multigrid cycle. Beginning at level 0 with a zero initial solution, the restriction operator is applied three times to produce a coarse approximation at level 3. After this coarse-level system is solved, the results are interpolated to level 2 with the prolongation operator. A defect equation is established at level 2, and then restricted back to level 3. This processes is repeated at finer and finer levels until level 0 is reached. After returning to level 0, V-cycles are applied (as necessary) to further reduce the residual. Prolongation and restriction operators for the original equations are applied to the dashed steps while operators for the defect equations are applied to the rest of the steps.

threshold, δ^l, at each level is set to the average edge length at that level multiplied by $r^{-1/3}$. Such thresholds improve the isotropy of the coarsening steps. The extraction of a maximal δ-independent set is implemented using a sweep algorithm. Once a δ-independent vertex set has been found, they are elevated to the next coarser level and connectivity among them is set up. There should be an edge between v_i^{l+1} and v_j^{l+1} in this coarser graph if v_j^l is within the 2-ring neighborhood of v_i^l. Here v_i^{l+1} and v_j^{l+1} are the corresponding vertices of v_i^l and v_j^l in the coarser graph, respectively.

Unlike most previous multigrid techniques [76, 24, 25] for mesh processing which construct a hierarchy using mesh simplification steps (such as edge contraction) as well as elevating all constrained vertices to the coarsest level, we completely avoid these steps. Consequently, we avoid the overhead for maintaining valid meshes and our coarsened graphs have fewer vertices. In comparison to the fast MIS hierarchy proposed in [25], our hierarchy adopts an additional distance threshold and a simplified edge construction scheme for the coarsened graph. As a result, our graph-based hierarchy can achieve aggressive coarsening with a fast decay rate to facilitate fast convergence of the multigrid algorithm. Meanwhile, constructing the hierarchy itself can be made very efficient due to its simplicity.

7.5 Our Multigrid Algorithm

We adopt the full multigrid algorithm discussed in Section 7.1.1 to solve the equations we adopt in Section 7.2 for the two passes during mesh deformation. During the first pass, we adopt the Laplace equation in

(7.1) to compute harmonic fields over the mesh surface to interpolate both rotation and scaling constraints. During the second pass, we adopt the discrete Poisson equation in (7.8) to compute new vertex coordinates. Since the Laplace equation is a special case of the Poisson equation and the same discrete Laplacian operator in (7.8) can be used for discretizing (7.1), in the following, we will focus our discussion on the more general equation in (7.8).

7.5.1 Solver for the Coarsest Level

At the coarsest level of our graph hierarchy, we obtain an accurate solution using a direct solver for sparse linear systems. We use TAUCS [3, 93] which performs a sparse Cholesky factorization followed by back substitution. The coarsest level in a hierarchy typically has between 2000 and 2500 vertices which correspond to three times as many unknowns and TAUCS can solve this many unknowns very quickly.

7.5.2 Smoothing Operator

As mentioned, we directly adopt (7.7) as the smoothing operator at each level. This implies that we follow a sequential order to update the coordinates of all vertices in the same level. When we use (7.7) to update the coordinates of $\mathbf{v}_i$, some coordinates on the right hand side of (7.7) might already have been updated.

7.5.3 Prolongation/Restriction Operators

Since the full multigrid algorithm needs to restrict both the original equation and the defect equation to all intermediate levels of the hierarchy, we derive two distinct pairs of prolongation and restriction operators tailored for each of the equations to improve performance. As shown in Fig. 7.4, operators for the original equation are applied when there is no initial solution available; and operators for the defect equation are applied otherwise.

Let us start with the original equation. As we have seen, at the finest level, (7.8) becomes (7.11) once constraints are taken into account. It turns out that the restricted versions of (7.11) at coarser levels of the hierarchy will take the same general form of (7.11), but with different weights and constants. This will become clearer later in this section. Let the coordinates of a vertex, $\mathbf{v}_i^l$, at level l of the hierarchy be denoted as $\mathbf{x}_i^l$. If this vertex is elevated to the next coarser level, we simply increment its superscript. Then (7.11) can be rewritten in the new notation as follows,

$$\sum_{j \in N^l(i)} w_{ji}^l (\mathbf{x}_i^l - \mathbf{x}_j^l - \mathbf{d}_{ji}^l) + \alpha_i^l \mathbf{x}_i^l = \beta_i^l. \tag{7.12}$$

At the finest level ($l = 1$), $N^1(i) = N(i) \backslash C(i)$, α_i^1 and β_i^1 are respectively the same as α_i and β_i in (7.11).

We derive a distinct prolongation operator for (7.12) at each vertex. If a vertex is elevated to level $l+1$ during graph coarsening, its prolongation operator is simply the identity operator. Otherwise, suppose vertex $\mathbf{v}_i^l$ is retained at level l. The 1-ring neighbors of $\mathbf{v}_i^l$ can be further divided into two non-overlapping subsets which are indexed by $R^l(i)$ and $K^l(i)$, respectively. $R^l(i)$ indexes the subset of neighbors that are elevated to level $l+1$ while $K^l(i)$ indexes those that are retained at level l. Note that $R^l(i)$ is not empty according to the graph coarsening discussed in Section 7.4. Thus, (7.12) can be rearranged as follows,

$$\sum_{j \in R^l(i)} w_{ji}^l (\mathbf{x}_i^l - \mathbf{x}_j^l - \mathbf{d}_{ji}^l) + \sum_{j \in K^l(i)} w_{ji}^l (\mathbf{x}_i^l - \mathbf{x}_j^l - \mathbf{d}_{ji}^l) + \alpha_i^l \mathbf{x}_i^l = \beta_i^l. \tag{7.13}$$

A prolongation operator at $\mathbf{v}_i^l$ should approximate its coordinates only through a function of those neighbors that have been elevated to level $l+1$. To achieve this goal with minimal "damage", we simply remove the edges between $\mathbf{v}_i^l$ and those neighbors indexed by $K^l(i)$. Such pruning can be done by setting the weights of these edges to zero. This gives rise to the following prolongation operator at $\mathbf{v}_i^l$ for the original equation,

$$\mathbf{x}_i^l = \frac{\sum_{j \in R^l(i)} w_{ji}^l (\mathbf{x}_j^{l+1} + \mathbf{d}_{ji}^l) + \beta_i^l}{\sum_{j \in R^l(i)} w_{ji}^l + \alpha_i^l}. \tag{7.14}$$

Suppose $\mathbf{v}_j^l$ is one of $\mathbf{v}_i^l$'s elevated neighbors, therefore, $j \in R^l(i)$. The original equation at $\mathbf{v}_j^l$ involves $\mathbf{x}_i^l$. If we substitute the right hand side of (7.14) into that equation, we can successfully eliminate $\mathbf{x}_i^l$. Similarly, we can eliminate $\mathbf{x}_i^l$ from all original equations. At an even larger scale, we can use prolongation operators similar to (7.14) to eliminate all vertices that have been retained at level l from all equations corresponding to those vertices that have been elevated to level $l+1$. The resulting equations only involve the vertices at level $l+1$ of the graph hierarchy, and they become the so-called restricted original equations at level $l+1$. More concretely, by substituting those prolongation operators at $\mathbf{v}_j^l$'s retained 1-ring neighbors into $\mathbf{v}_j^l$'s original equation at level l, we obtain the equation for $\mathbf{v}_j^{l+1}$ in the coarser level,

$$\begin{aligned} \sum_{i \in R^l(j)} w_{ij}^l (\mathbf{x}_j^{l+1} - \mathbf{x}_i^{l+1} - \mathbf{d}_{ij}^l) \quad & + \\ \sum_{i \in K^l(j)} w_{ij}^l \left(\mathbf{x}_j^{l+1} - \left(\frac{\sum_{k \in R^l(i)} w_{ki}^l (\mathbf{x}_k^{l+1} + \mathbf{d}_{ki}^l) + \beta_i^l}{\sum_{k \in R^l(i)} w_{ki}^l + \alpha_i^l} \right) - \mathbf{d}_{ij}^l \right) \quad & + \\ \alpha_j^l \mathbf{x}_j^{l+1} \quad = \quad & \beta_j^l, \end{aligned} \tag{7.15}$$

where the inner sum iterates over the elevated neighbors of each $\mathbf{v}_i^l$ as illustrated in Fig. 7.5. Note that

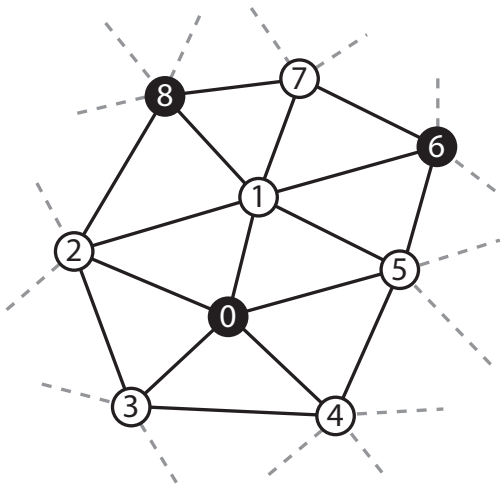


Figure 7.5: In this example, the black vertices will be raised to level $l+1$ while the white vertices are retained at the current level l. Since $R^l(1) = \{0, 6, 8\}$, the prolongation operator (7.14) for $\mathbf{v}_1^l$ will involve the terms $\mathbf{v}_0^{l+1}, \mathbf{v}_6^{l+1}$ and $\mathbf{v}_8^{l+1}$. The restriction operator (7.15) relates $\mathbf{v}_0^{l+1}$ to the other raised vertices within its two-ring. In this case, the following (j, i, k) paths contribute to the sum, $\{(0,1,0),(0,2,0),(0,3,0),(0,4,0),(0,5,0)\}$ and $\{(0,1,6),(0,1,8),(0,2,8),(0,5,6)\}$.

δ-independent coarsening allows for elevated 1-ring neighbors of an elevated vertex $\mathbf{v}_j^l$. Interestingly, these new equations at level $l+1$ can still be arranged to follow the general form given in (7.12). However, the weights and constants have been updated. The formulations of the updated weights and constants can be found in the Appendix C.

Importantly, variable substitutions do not make the restricted equations denser. The equation for every vertex at level $l+1$ still involves only the 1-ring neighbors of that vertex. This is because both the prolongation operator and variable substitutions only involve 1-ring neighborhoods. Therefore, two vertices appearing in the same resulting equation would be in each other's 2-ring neighborhood at level l. According to our graph coarsening, such pairs of vertices would be in each other's 1-ring neighborhood at level $l+1$.

Now let us proceed to the defect equation. To derive the defect equation of (7.12), we need to replace every $\mathbf{x}_i^l$ with $\mathbf{x}_i^l + \Delta\mathbf{x}_i^l$ where $\mathbf{x}_i^l$ is fixed and $\Delta\mathbf{x}_i^l$ becomes the unknown. Thus, the defect equation of (7.12) at $\mathbf{v}_i^l$ is as follows.

$$\sum_{j \in N^l(i)} w_{ji}^l (\Delta\mathbf{x}_i^l - \Delta\mathbf{x}_j^l) + \alpha_i^l \Delta\mathbf{x}_i^l = \zeta_i^l, \tag{7.16}$$

where $\zeta_i^l = \beta_i^l - \alpha_i^l \mathbf{x}_i^l - \sum_{j \in N^l(i)} w_{ji}^l (\mathbf{x}_i^l - \mathbf{x}_j^l - \mathbf{d}_{ji}^l)$. Note that the defect equation is similar to the original equation in (7.12) except that there are no $\mathbf{d}_{ji}^l$'s on the left hand side and the constant on the right hand side is different.

We derive a separate pair of prolongation and restriction operators tailored for the defect equation, especially for vertices retained at level l. For a retained vertex $\mathbf{v}_i^l$, we expect the residual solution, $\Delta\mathbf{x}_j^l (j \in$

$K^l(i)$), in its neighborhood to be small once a good initial solution has been obtained. Instead of pruning entire edges, we would like to keep the initial solution but set the residual coordinates of those neighbors indexed by $K^l(i)$ to be zero. Thus, the prolongation operator at $\mathbf{v}_i^l$ for the defect equation is formulated as

$$\Delta \mathbf{x}_i^l = \frac{\sum_{j \in R^l(i)} w_{ji}^l \Delta \mathbf{x}_j^{l+1} + \zeta_i^l}{\sum_{j \in N^l(i)} w_{ji}^l + \alpha_i^l}. \tag{7.17}$$

A corresponding restriction operator similar to (7.15) can also be obtained.

As shown above, at every level we have a distinct pair of prolongation and restriction operators at every vertex. They are derived using a weighted graph model and algebraic manipulations of the equations. During such derivation, they are not restricted to be such linear operators as the $\mathbf{P}$ and $\mathbf{R}$ matrices in (7.10). Intuitively, instead of producing a smooth interpolation from coarser level vertices, our prolongation operators actually use original equations, such as (7.5), to generate more accurate estimations for a retained vertex from its elevated 1-ring neighbors. Therefore, they can make the restricted equations at different levels more consistent with each other. As a result, solutions at a coarser level only need to have minimal revision at a finer level after being interpolated using the prolongation operators. The weights in the restricted equations at each level are computed directly on the graph hierarchy instead of using sparse matrix multiplications. This is because searching for neighbors utilizing graph connectivity can be performed more efficiently on a graph data structure.

7.6 Experimental Results

In our implementation of the multigrid solver presented in the previous sections, we always take zero as the initial guess, adopt the full multigrid algorithm with V-cycles, and apply two pre-smoothing and two post-smoothing steps. Our graph coarsening strategy maintains a healthy ratio of the number of vertices between adjacent levels, which is around 5, and the sparsity structures of the linear systems are similar at different levels. For example, the average number of nonzero entries per row is respectively 12.4, 10.8, 9.8, 10.7 and 11.0 at the five levels of the hierarchy for the LUCY model. When the desired relative residual is 1e-5 or lower, performance can be further improved by 30% if every V-cycle is followed by two iterations of preconditioned conjugate gradient.

We have tested our implementation on meshes with increasing complexity. Table 7.1 summarizes the performance and scalability of different solvers applied to the equations described in Section 7.2.2. All the meshes, except the last one, reported in Table 7.1 are embedded in a volume graph. Therefore, the number of free vertices include both mesh vertices and additional volume graph vertices. The original surface

		SPRING	DINO	CAMEL	FELINE	FEMALE	LUCY	DRAGON
#Free Vertices		24,188	43,494	99,588	181,292	415,619	822,204	3,447,861
UMFPACK	Factor (s)	1.63	2.72	20.59	37.29	113.11	n/a	n/a
	Substitute (s)	0.16	0.26	1.04	1.95	5.00	n/a	n/a
	Memory (MB)	52	70	398	710	1,838	>2 GB	>2 GB
CHOLMOD	Factor (s)	0.43	0.83	5.48	12.20	31.9	69.32	n/a
	Substitute (s)	0.03	0.05	0.15	0.30	0.78	1.36	n/a
	Memory (MB)	26	35	139	292	695	1,311	>2 GB
TAUCS	Factor (s)	0.60	1.04	4.70	10.46	25.90	57.65	n/a
	Substitute (s)	0.09	0.16	0.57	1.197	2.63	5.35	n/a
	Memory (MB)	25	41	139	277	643	1,190	>2 GB
Trilinos ML	Setup (s)	0.15	0.34	0.57	1.06	2.63	4.87	12.60
	Solve (s)	0.57	2.19	5.37	9.15	24.00	47.22	148.80
	Memory (MB)	15	21	52	87	200	388	1,080
Our Multigrid	Setup (s)	0.06	0.16	0.13	0.24	0.58	0.94	2.64
	Solve (s)	0.17	0.34	0.76	1.75	4.07	8.10	36.94
	Memory (MB)	10	16	31	56	119	232	740
	#Levels / #V-cycles	3 / 3	4 / 4	4 / 4	5 / 6	5 / 6	6 / 7	9 / 8

Table 7.1: UMFPACK [1] produces LU factorizations for general sparse matrices. CHOLMOD [2] and TAUCS [3] factor sparse Cholesky matrices and are among the fastest direct solvers for this problem. Trilinos ML [4] denotes the multilevel preconditioner ML used via the Trilinos AztecOO interface. Factorization and back-substitution times are reported for the direct solvers while timing for hierarchy construction and iteration to 1e-5 relative residual are recorded for the multilevel solvers. Peak memory consumption is recorded for Trilinos ML and our solver. For UMFPACK, CHOLMOD, and TAUCS, the reported memory cost is for the factors alone. While the system is solved for each of the x,y, and z coordinates, only one factorization and three back-substitutions are required of the direct solvers. Likewise, the multilevel hierarchy is created once and reused. Data has been excluded in tests where memory use exceeded hardware limits. Timing data was collected on a pair of comparably equipped high-end uniprocessors ($\sim$ Pentium IV 3.8GHz) with 2GB physical memory. In every test, our solver exhibits the best performance and memory efficiency.

meshes have respectively 7800, 14050, 21887, 49864, 132736, 262909, and 3609455 vertices. Here we measure performance using the total time a solver needs to decrease the relative residual down to a given precision. This is a better performance metric than the number of cycles or iterations because the computational cost of each cycle or iteration differs among different solvers. In every test, our multigrid solver exhibits the best performance and memory efficiency. Direct factorization methods are much less scalable than multigrid algorithms in both computational and memory costs. Table 7.1 reveals a super-linear relationship in both factorization time and memory cost (the size of the resultant factors) with increasing mesh complexity. Even TAUCS and CHOLMOD, both highly efficient sparse Cholesky codes, exceed available memory in the largest test. In contrast, multilevel algorithms exhibit linear scaling in both time and memory costs, making them a desired option for multi-million vertex meshes. Such comparisons indicate that multilevel solvers are a better choice for mesh editing operations that result in a new coefficient matrix which otherwise needs to be factorized using a direct solver.

In applications where the matrix remains constant, factorization is a one-time cost and only back-

		SPRING	DINO	CAMEL	FELINE	FEMALE	LUCY	DRAGON
PCG	Residual < 1e-3	0.94	2.20	10.31	19.52	88.75	167.50	n/a
Trilinos	Residual < 1e-3	0.36	0.90	3.36	5.94	14.22	27.63	85.86
Our Multigrid	Residual < 1e-3	0.11	0.16	0.29	0.50	1.23	2.23	8.14

Table 7.2: Timing data for three iterative solvers to reach 1e-3 relative residual (in seconds). PCG denotes preconditioned conjugate gradient with incomplete Cholesky decomposition. PCG did not converge to this precision within a reasonable amount of time on the largest mesh. Our multigrid solver can reach such an intermediate level of precision almost one order of magnitude faster than Trilinos. It is also competitive with the back-substitution times of direct solvers on large meshes (Table 7.1). The ability to quickly generate good approximate solutions is especially important when interactivity is demanded.

substitutions are necessary to solve new systems. For example, when one manipulates relatively small regions of interest on a mesh, back-substitutions are fast and direct solvers are clearly the favorable choice. However, are direct solvers always advantageous here? Unfortunately not because the cost of back-substitution is proportional to the size of the factors which tend to be much denser than the original sparse matrix. For a sufficiently large mesh, solving via multigrid can be faster than even a single back-substitution. According to Tables 7.1 and 7.2, back-substitution of three coordinates in TAUCS and UMFPACK for the CAMEL model already exceeds the time required by our multigrid solver to reach an intermediate approximate solution (relative residual 1e-3).

Meanwhile, we wish to minimize the cost of constructing the multigrid hierarchy. As demonstrated in Table 7.1, our simple coarsening strategy can be implemented efficiently. Moreover, numerical accuracy and visual quality are not always consistent, i.e. at the same residual, the visual acceptability of the output of different solvers will vary. As shown in Table 7.2, our algorithm effectively distributes errors and rapidly produces approximate solutions (< 1e-3 relative residual) which may be sufficient for many applications. In fact, all of the visual results reported in this work were generated at this level of numerical accuracy.

We have made an effort to include factorization methods that are both representative of those in common use (UMFPACK) and those with the best performance (CHOLMOD,TAUCS). In the latter case, we do not claim that our choices are optimal. Indeed, for a particular linear system, other solvers may surpass our selections in factorization time, memory cost, or both. Nevertheless, more comprehensive comparisons [94] suggest that CHOLMOD and TAUCS are among the best freely available sparse Cholesky codes. Likewise, Trilinos ML, developed by Sandia National Laboratories, is a competitive representative for algebraic multilevel (AMG) methods. It is a fully optimized code with a very effective multilevel preconditioner.

Lastly, we note that the number of free vertices cannot be our only metric as the particular mesh structure may affect solver performance. This point is evidenced by the relatively high figures for all three factorization methods on the CAMEL model in Table 7.1. The density of the underlying graph must also be considered. For example, most volume graph vertices have valence ≥ 14, while a surface mesh has few, if any, such

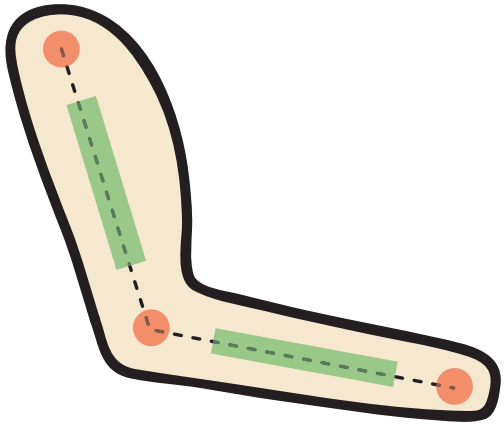


Figure 7.6: This figure illustrates how MOCAP data can be used to establish mesh deformation constraints. In the first pass, rotation constraints are applied to all volume graph vertices within the green regions, thereby maintaining bone rigidity. Positions for all graph vertices are then solved in the second pass, using vertices within the orange regions as constraints.

vertices.

7.7 Applications

Mesh deformation has a number of applications. We briefly discuss two of them here. First, intuitive mesh deformation is a powerful modeling tool. We have implemented a simple user interface for this purpose. During an interactive session, the user only needs to manipulate one handle at a time and the rotation field is obtained using our multigrid solver on the fly every time the handle is changed. To demonstrate that our fast multigrid algorithm can be integrated into a general mesh editing environment, we have implemented a few simple mesh editing tools, such as cutting, merging, local remeshing, surface curve sketching and insertion, and tested interleaving deformation with these operations. As an example, we created a composite model (Fig. 7.8) from four individual meshes each of which is deformed multiple times. Between successive deformation operations, local remeshing was sometimes performed to avoid triangles with extreme aspect ratios. The performance of our multigrid solver made it possible to quickly construct the hierarchy and obtain a solution every time we have performed remeshing or merging. In previous work [80, 82], large meshes were first simplified before they were deformed. Such a scheme would not be appropriate for extremely large deformations. For instance, the DRAGON model in Fig. 7.8 has been stretched more than twice to form the spiral shape around the LUCY model. Without applying mesh subdivision to increase the number of vertices, it would not have been possible to perform such a large-scale deformation.

Second, with a powerful mesh deformation technique, it has become practical to create interesting mesh animations from only one single base mesh. We have conducted experiments to use our solver to animate a

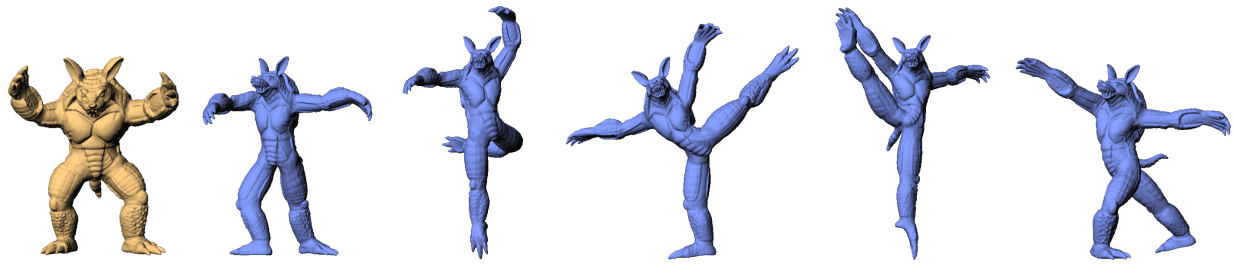


Figure 7.7: This figure shows the initial ARMADILLO mesh followed by a few frames from the ballet sequence. The entire volume graph has 525K free vertices and the running time is 2.88 seconds/frame. Combining the 20 rotation constraints at each vertex requires a non-negligible portion of the per frame time. These results were generated by our solver at an accuracy of $<$ 1e-3 relative residual.

mesh with a given MOCAP animation (Figures 7.1 & 7.7). We begin by constructing a volume graph for the base mesh in its original pose. We then select volume graph vertices within a cylindrical region along each bone of the skeleton used for the MOCAP data. Using the data from the animation, all vertices within each of these regions follow a single rotation constraint during the first pass. Likewise, volume graph vertices contained within a spherical region centered at each joint provide position constraints during the second pass. These regions are illustrated in Fig. 7.6. Although the rotation constraints are changing from frame to frame, they are always applied to the same subset of vertices throughout an entire animation. Therefore, the scalar harmonic fields for interpolating the rotation constraints need to be computed only once in a preprocessing step. Most often, the initial pose of the MOCAP data differs significantly from the pose of the base mesh. An initial deformation that transforms the base mesh from its original pose to the initial pose of the MOCAP data should be performed.

7.8 Conclusions

We have developed an efficient multigrid solver suited for fast mesh deformation. Our solver maintains a significant advantage over other multigrid techniques in both hierarchy construction and solution time. It can also trade off accuracy for speed to achieve greater interactivity. These properties are desired in situations where the existence of other operations preclude the use of extensive precomputation as such results will be frequently invalidated. We have applied our solver to static mesh editing as well as mesh animation. Because of the unstructured nature of the graphs we use, a GPU implementation of the smoothing operator did not prove any faster than on the CPU. Nevertheless, multigrid methods are parallelizable. With the advent of multicore CPUs, our solver can be made multiple times faster.

Although our multigrid solver can achieve a relative residual of 1e-7 on all examples given in this chapter, we do not currently have a convergence proof. However, our solver has a great potential for further

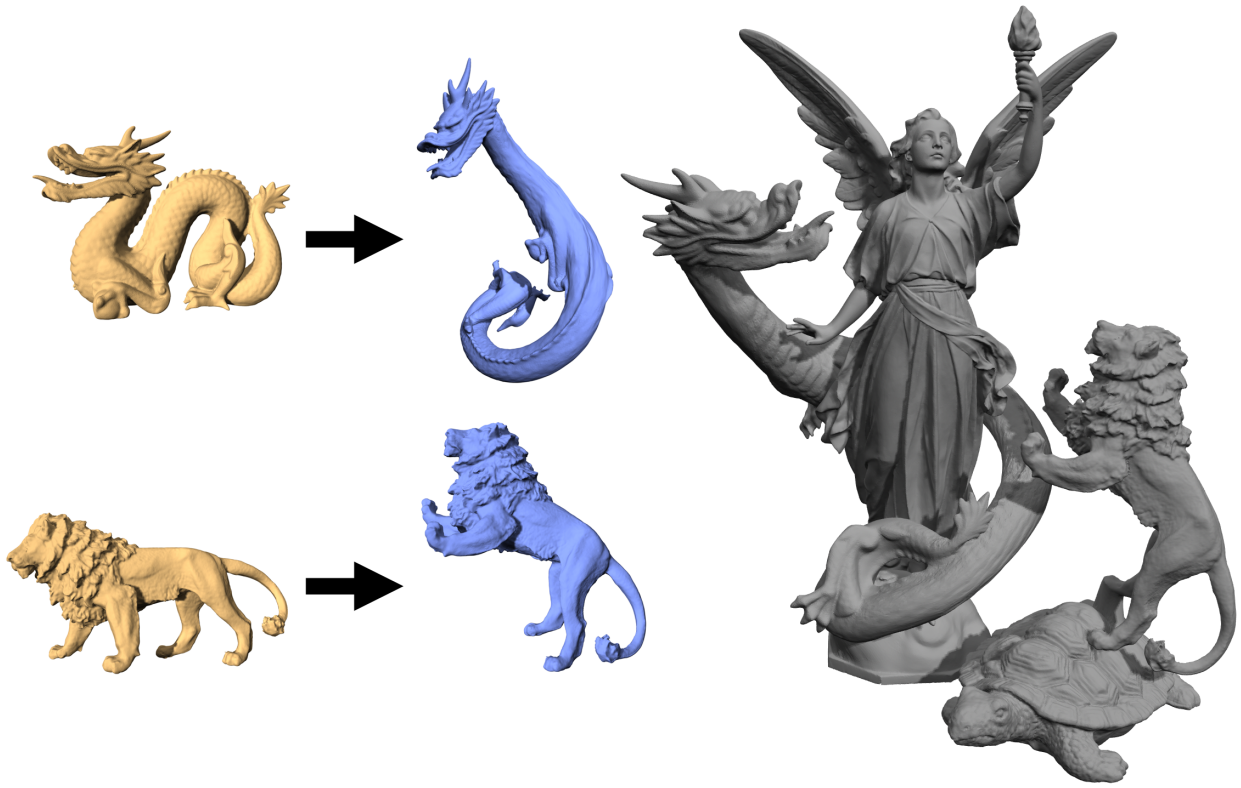


Figure 7.8: Four meshes, with several hundred thousand vertices each, are edited and then merged to form a large statue. Each of the remeshing and merging operations used during this session gives rise to a new linear system. Under these circumstances, the setup cost of direct factorization methods become prohibitively expensive.

optimization. In fact, its performance has been much improved by interleaving V-cycles and preconditioned conjugate gradient. Our solver can be potentially extended to other mesh-related problems, including surface parameterization, fairing and remeshing. One limitation is that the topological Laplacian with symmetric weights adopted in this work prevents a straightforward extension to problems where the Laplacian has nonsymmetric weights. Nonetheless, our weighted graph based methodology will still be useful in deriving effective prolongation and restriction operators for other linear systems defined on unstructured meshes.

Appendix A

Variational Gradient

In this section, we provide an intuitive derivation of (3.3) from (3.2) in a 2D space. The derivation for a 3D space can be obtained similarly. The smoke boundary is first discretized into a finite set of points. Consider one of the points $\mathbf{x}_i$ shown in Fig. A.1. We need to perturb the position of $\mathbf{x}_i$ to reduce the integral in (3.2). Obviously, $\mathbf{x}_i$ should move further into the interior of the target object. If $\mathbf{x}_i$ moves along the local normal at $\mathbf{x}_i$ by an infinitesimal distance, (3.2) is reduced by an amount equal to the area enclosed by the polygon $\mathbf{x}_{i-1}\mathbf{x}_i\mathbf{x}_{i+1}\mathbf{x}_i^N$ where $\mathbf{x}_i^N$ represents the new location of $\mathbf{x}_i$. If $\mathbf{x}_i$ moves along some other arbitrary direction to $\mathbf{x}_i'$ by the same infinitesimal distance, (3.2) is reduced by an amount equal to the area enclosed by the polygon $\mathbf{x}_{i-1}\mathbf{x}_i\mathbf{x}_{i+1}\mathbf{x}_i'$. Since $\mathbf{x}_i\mathbf{x}_i^N$ is perpendicular to the local boundary, when the lengths of $\mathbf{x}_{i-1}\mathbf{x}_i$ and $\mathbf{x}_i\mathbf{x}_{i+1}$ become sufficiently small, the area of $\mathbf{x}_{i-1}\mathbf{x}_i\mathbf{x}_{i+1}\mathbf{x}_i^N$ is guaranteed to be larger than the area of $\mathbf{x}_{i-1}\mathbf{x}_i\mathbf{x}_{i+1}\mathbf{x}_i'$. That is, (3.2) can be decreased most quickly by moving $\mathbf{x}_i$ along the local normal of the smoke boundary. This also holds for all other points on the same boundary. By increasing the number of points in the boundary discretization, the distance between two adjacent points can be arbitrarily close to zero. By definition [34], the variational derivative of the functional in (3.2) with respect to the geometry of the smoke boundary exists. The variational derivative with respect to a specific boundary point is the unit normal vector at that point multiplied by an appropriate sign given in (3.3).

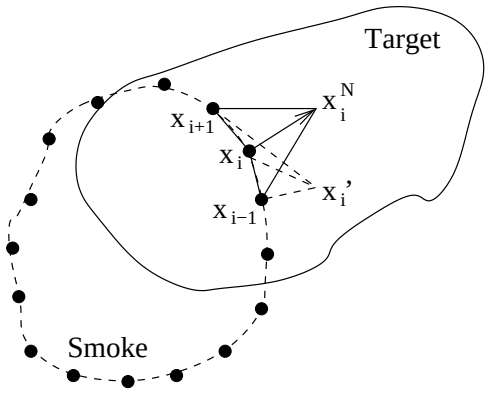


Figure A.1: Moving points on the smoke boundary along their respective normal directions can reduce the shape discrepancy between the smoke region and the target object most quickly.

Appendix B

An Equivalent Condition for $\nabla \cdot \mathbf{u} = 0$

We show that, for a bounded workspace Ω without sources and sinks, $\nabla(\nabla \cdot \mathbf{u}) = 0$ everywhere is equivalent to $\nabla \cdot \mathbf{u} = 0$ everywhere. Without loss of generality, suppose there exists a point $\mathbf{x_0} \in \Omega$ such that $\nabla \cdot \mathbf{u}(\mathbf{x_0}) > 0$. Since Ω is bounded,

$$\int_{\Omega} (\nabla \cdot \mathbf{u}) d\mathbf{x} = 0.$$

Therefore, there must be another point $\mathbf{x_1} \in \Omega$ such that $\nabla \cdot \mathbf{u}(\mathbf{x_1}) < 0$. Consider an arbitrary path between $\mathbf{x_0}$ and $\mathbf{x_1}$. There must exist a point $\mathbf{x_m}$ on that path such that $\nabla(\nabla \cdot \mathbf{u}(\mathbf{x_m})) \neq 0$. Otherwise, $\nabla \cdot \mathbf{u}(\mathbf{x_0}) = \nabla \cdot \mathbf{u}(\mathbf{x_1})$. Thus, $\nabla \cdot \mathbf{u}$ must be zero everywhere.

Appendix C

Relationships between Weights and Constants at Adjacent Levels

The restricted original equation of $\mathbf{v}_j^{l+1}$ at level $l+1$ is formulated as

$$\sum_{k \in N^{l+1}(j)} w_{kj}^{l+1}(\mathbf{x}_j^{l+1} - \mathbf{x}_k^{l+1} - \mathbf{d}_{kj}^{l+1}) + \alpha_j^{l+1}\mathbf{x}_j^{l+1} = \beta_j^{l+1}. \tag{C.1}$$

Suppose $k \in N^{l+1}(j)$ and $\mathbf{v}_k^{l+1}$ is a 1-ring neighbor of $\mathbf{v}_j^{l+1}$ at level $l+1$. We also assume that $|K^l(j) \cap K^l(k)| = m$, which means that there are m indirect paths between $\mathbf{v}_j^l$ and $\mathbf{v}_k^l$ at level l (Fig. 7.5 defines such paths). Then, we can obtain the following relationships between weights and constants at the two levels:

$$w_{kj}^{l+1} = \Psi(k,j)w_{kj}^l + \sum_{i \in K^l(j) \cap K^l(k)} \frac{w_{ki}^l w_{ij}^l}{Z_i^l}, \tag{C.2}$$

$$\mathbf{d}_{kj}^{l+1} = \frac{1}{w_{kj}^{l+1}} \left(\Psi(k,j)w_{kj}^l \mathbf{d}_{kj}^l + \sum_{i \in K^l(j) \cap K^l(k)} \frac{w_{ki}^l w_{ij}^l (\mathbf{d}_{ki}^l + \mathbf{d}_{ij}^l)}{Z_i^l} \right) \tag{C.3}$$

$$\alpha_j^{l+1} = \alpha_j^l + \sum_{i \in K^l(j)} \frac{w_{ij}^l \alpha_i^l}{Z_i^l}, \tag{C.4}$$

$$\beta_j^{l+1} = \beta_j^l + \sum_{i \in K^l(j)} \frac{w_{ij}^l \left(\beta_i^l + \alpha_i^l \mathbf{d}_{ij}^l\right)}{Z_i^l}, \tag{C.5}$$

where $Z_i^l = \sum_{s \in R^l(i)} w_{si}^l + \alpha_i^l$, and $\Psi(k,j)$ is one when $\mathbf{v}_k^l$ happens to be a 1-ring neighbor of $\mathbf{v}_j^l$ and zero otherwise.

References

[1] T. A. Davis, *UMFPACK Version 4.4 User Guide*, Tech. Rep. TR-04-003, University of Florida (2005).

[2] T. A. Davis, *User Guide for CHOLMOD*, Tech. Rep., University of Florida (2006).

[3] S. Toledo, V. Rotkin, and D. Chen, *Taucs:a library of sparse linear solvers. version 2.2*, Tech. Rep., Tel-Aviv University (2003).

[4] M. A. Heroux and J. M. Willenbring, *Trilinos Users Guide*, Tech. Rep. SAND2003-2952, Sandia National Laboratories (2003).

[5] J. Hong and C. hun Kim, *Controlling fluid animation with geometric potential*, Computer Animation & Virtual Worlds **15**, 147 (2004).

[6] L. Shi and Y. Yu, *Controllable Smoke Animation with Guiding Objects*, ACM Transactions on Graphics **24**, 140 (2005).

[7] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or, *Linear Rotation-Invariant Coordinates for Meshes*, ACM Transactions on Graphics **24** (2005).

[8] A. Kapler, *Evolution of a vfx voxel tool*, in *SIGGRAPH 2002 Sketches & Applications, Conference Abstracts and Applications* (2002), p. 179.

[9] J. Stam, *Multi-scale stochastic modeling of complex natural phenomena*, Ph.D. thesis, University of Toronto (1995).

[10] N. Foster and D. Metaxas, *Controlling fluid animation*, in *Proceedings of Computer Graphics International* (1997), pp. 178–188.

[11] N. Foster and R. Fedkiw, *Practical animation of liquids*, in *SIGGRAPH 2001 Conference Proceedings* (2001), pp. 23–30.

[12] A. Lamorlette and N. Foster, *Structural modeling of flames for a production environment*, in *SIGGRAPH 02 Conference Proceedings* (2002), pp. 729–735.

[13] Y. Yu and L. Shi, *Object modeling and animation with smoke*, Tech. Rep. UIUCDCS-R-2002-2262, Computer Science, University of Illinois at Urbana-Champaign (2002), http://www.cs.uiuc.edu/Dienst/UI/2.0/Describe/ncstrl.uiuc_cs/UIUCDCS-R-2002-2262.

[14] R. Fattal and D. Lischinski, *Target-driven smoke animation*, ACM Transactions on Graphics **23**, 439 (2004).

[15] A. Treuille, A. McNamara, Z. Popovic, and J. Stam, *Keyframe Control of Smoke Simulations*, ACM Trans. Graphics **22**, 716 (2003).

[16] A. McNamara, A. Treuille, Z. Popovic, and J. Stam, *Fluid control using the adjoint method*, ACM Transaction on Graphics **23**, 447 (2004).

[17] F. Pighin, J. M. Cohen, and M. Shah, *Modeling and editing flows using advected radial basis functions*, in *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (ACM Press, New York, NY, USA, 2004), pp. 223–232, ISBN 3-905673-14-2.

[18] V. Mihalef, D. Metaxas, and M. Sussman, *Animation and control of breaking waves*, in *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (ACM Press, New York, NY, USA, 2004), pp. 315–324, ISBN 3-905673-14-2.

[19] J. Wejchert and D. Haumann, *Animation aerodynamics*, in *Proceedings of SIGGRAPH* (1991), pp. 19–22.

[20] S. Chenney, *Flow tiles*, in *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (ACM Press, New York, NY, USA, 2004), pp. 233–242, ISBN 3-905673-14-2.

[21] X. Wei, Y. Zhao, Z. Fan, W. Li, S. Yoakum-Stover, and A. Kaufman, *Blowing in the wind*, in *ACM Symposium on Computer Animation* (2003).

[22] A. Witkin and M. Kass, *Spacetime constraints*, in *SIGGRAPH 88 Proceedings* (1988), pp. 159–168.

[23] J. Stam, *Flows on Surfaces of Arbitrary Topology*, ACM Transactions on Graphics **22**, 724 (2003).

[24] N. Ray and B. Levy, *Hierarchical least squares conformal map*, in *Proceedings of Pacific Graphics* (2003).

[25] B. Aksoylu, A. Khodakovsky, and P. Schröder, *Multilevel Solvers for Unstructured Surface Meshes*, SIAM Journal on Scientific Computing **26**, 1146 (2005).

[26] G. Yngve, J. O'Brien, and J. Hodgins, *Animating explosions*, in *SIGGRAPH 2000 Conference Proceedings* (2000), pp. 29–36.

[27] N. Foster and D. Metaxas, *Modeling the motion of a hot, turbulent gas*, in *SIGGRAPH 97 Conference Proceedings* (1997), pp. 181–188.

[28] J. Stam, *Stable fluids*, in *SIGGRAPH 99 Conference Proceedings* (1999), pp. 121–128.

[29] R. Fedkiw, J. Stam, and H. Jensen, *Visual simulation of smoke*, in *SIGGRAPH 01 Conference Proceedings* (2001), pp. 15–22.

[30] A. Staniforth and J. Cote, *Semi-lagrangian integration schemes for atmospheric models: A review*, Monthly Weather Review **119**, 2206 (1991).

[31] G. Golub and C. Loan, *Matrix Computations* (The Johns Hopkins University Press, 1996).

[32] J. Monaghan, *Smoothed Particle Hydrodynamics*, Annual Review of Astronomy and Astrophysics **30**, 543 (1992).

[33] J. Monaghan, *Simulating free surface flows with SPH*, Journal of Computational Physics **110**, 399 (1994).

[34] I. Gelfand and S. Fomin, *Calculus of Variations* (Prentice-Hall, 1963).

[35] K. Museth, D. Breen, R. Whitaker, and A. Barr, *Level Set Surface Editing Operators*, ACM Transactions on Graphics **21**, 330 (2002).

[36] S. Osher and J. Sethian, *Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations*, Computational Physics **79**, 12 (1988).

[37] J. Sethian, *Level Set Methods and Fast Marching Methods* (Cambridge University Press, 1999).

[38] A. Chorin, *A Numerical Method for Solving Incompressible Viscous Flow Problems*, Computational Physics **2**, 12 (1967).

[39] G. Turk and J. O'Brien, *Shape transformation using variational implicit functions*, in *SIGGRAPH 99 Conference Proceedings* (1999), pp. 335–342.

[40] J. Carr, T. Mitchell, R. Beatson, J. Cherrie, W. Fright, B. McCallum, and T. Evans, *Reconstruction and representation of 3d objects with radial basis functions*, in *SIGGRAPH 2001 Conference Proceedings* (2001), pp. 67–76.

[41] G. Turk and J. O'Brien, *Modelling with Implicit Surfaces that Interpolate*, ACM Transactions on Graphics **21**, 855 (2002).

[42] J. Kajiya and B. von Herzen, *Ray Tracing Volume Densities*, Computer Graphics (SIGGRAPH 84 Proceedings) **18** (1984).

[43] S. Beauchemin and J. Barron, *The Computation of Optical Flow*, ACM Computing Surveys **27**, 433 (1995).

[44] M. Black and P. Anandan, *The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields*, Computer Vision and Image Understanding **63**, 75 (1996).

[45] N. Rasmussen, E. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw, *Directable photorealistic liquids*, in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2004).

[46] F. Bullo, *Stabilization of Relative Equilibria for Underactuated Systems on Riemannian Manifolds*, Automatica **36**, 1819 (2000).

[47] D. Enright, S. Marschner, and R. Fedkiw, *Animation and Rendering of Complex Water Surfaces*, ACM Transactions on Graphics **21**, 736 (2002).

[48] F. Losasso, F. Gibou, and R. Fedkiw, *Simulating Water and Smoke with an Octree Data Structure*, ACM Transactions on Graphics **23**, 455 (2004).

[49] T. Goktekin, A. W. Bargteil, and J. O'Brien, *A Method for Animating Viscoelastic Fluids*, ACM Trans. Graphics **23**, 461 (2004).

[50] M. Carlson, P. Mucha, and G. Turk, *Rigid Fluid: Animating the Interplay Between Rigid Bodies and Fluid*, ACM Trans. Graphics **23**, 375 (2004).

[51] S. Osher and R. Fedkiw, *Level set methods: an overview and some recent results*, Computational Physics **169** (2001).

[52] D. Enright, *Use of the particle level set method for enhanced resolution of free surface flows*, Ph.D. thesis, Stanford University (2002).

[53] N. Sumner, S. Hoon, W. Geiger, S. Marino, N. Rasmussen, and R. Fedkiw, *Melting and terminatrix*, in *SIGGRAPH 2003 Sketches & Applications, Conference Abstracts and Applications* (2003).

[54] F. Haugen, *PID Control of Dynamic Systems* (Tapir Forlag, 2004).

[55] D. Adalsteinsson and J. A. Sethian, *The fast construction of extension velocities in level set methods*, J. Comput. Phys. **148**, 2 (1999).

[56] R. Peyret and T. D. Taylor, *Computational methods for fluid flow*, Springer series in computational physics (Springer, 1985).

[57] R. Sumner and J. Popović, *Deformation Transfer for Triangle Meshes*, ACM Transactions on Graphics **23**, 397 (2004).

[58] L. Lucy, *A Numerical Approach to the Testing of the Fission Hypothesis*, Astronomical Journal **82**, 1013 (1977).

[59] R. Gingold and J. Monaghan, *Smoothed particle hydrodynamics: theory and application to non-spherical stars*, Monthly Notices of the Royal Astronomical Society **181**, 375 (1977).

[60] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard, *Interactive Control of Avatars Animated With Human Motion Data*, ACM TOG **21**, 491 (2002).

[61] L. Kovar, M. Gleicher, and F. Pighin, *Motion Graphs*, ACM TOG **21**, 473 (2002).

[62] O. Arikan, D. Forsyth, and J. O'Brien, *Motion Synthesis from Annotations*, ACM TOG **22**, 402 (2003).

[63] J. Kuffner and S. LaValle, *Rrt-connect: An efficient approach to single-query path planning*, in *Proc. IEEE Int'l Conf. on Robotics and Automation* (2000), pp. 995–1001.

[64] P. Cheng, Z. Shen, and S. LaValle, *RRT-based trajectory design for autonomous automobiles and spacecraft*, Archives of Control Sciences **11**, 167 (2001).

[65] D. James and K. Fatahalian, *Precomputing Interactive Dynamic Deformable Scenes*, ACM TOG **22**, 879 (2003).

[66] L. Ling, M. Damodaran, and R. Gay, *Aerodynamic force models for animating cloth motion in air flow*, The Visual Computer **12**, 84 (1996).

[67] A. Gersho and R. Gray, *Vector Quantization and Signal Compression* (Kluwer Academic Publishers, Norwell,Mass, 1992).

[68] M. Choi, J. Lee, and S. Shin, *Planning Biped Locomotion Using Motion Capture Data*, ACM TOG **22**, 182 (2003).

[69] *Fsqp software*, http://gachinese.com/aemdesign
/FSQPframe.htm, originally developed at the Institute for Systems Research, University of Maryland.

[70] R. Abraham, J. Marsden, and T. Ratiu, *Manifolds, Tensor Analysis, and Applications*, vol. 75 (Springer, 1988), applied Mathematical Sciences.

[71] K. Polthier and E. Preuss, *Variational approach to vector field decomposition*, in *Proc. Eurographics Workshop on Scientific Visualization* (2000).

[72] Y. Tong, S. Lombeyda, A. Hirani, and M. Desbrun, *Discrete Multiscale Vector Field Decomposition*, ACM Trans. Graphics **22**, 445 (2003).

[73] T. Sederberg and S. Parry, *Free-Form Deformation of Solid Geometric Models*, Computer Graphics(SIGGRAPH'86) **20**, 151 (1986).

[74] M. Botsch and L. Kobbelt, *Real-Time Shape Editing using Radial Basis Functions*, Computer Graphics Forum (Eurographics 2005) **24**, 611 (2005).

[75] D. Zorin, P. Schröder, and W. Sweldens, *Interactive mutiresolution mesh editing*, in *SIGGRAPH 97 Proceedings* (1997), pp. 259–268.

[76] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, *Interactive multi-resolution modeling on arbitrary meshes*, in *Proc. SIGGRAPH'98* (1998), pp. 105–114.

[77] I. Guskov, W. Sweldens, and P. Schröder, *Multiresolution signal processing for meshes*, in *Proc. SIGGRAPH'99* (1999), pp. 325–334.

[78] M. Alexa, *Differential coordinates for local mesh morphing and deformation*, The Visual Computer **19**, 105 (2003).

[79] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, *Laplacian surface editing*, in *Symposium of Geometry Processing* (2004).

[80] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, *Mesh Editing With Poisson-Based Gradient Field Manipulation*, ACM Transactions on Graphics (special issue for SIGGRAPH 2004) **23**, 641 (2004).

[81] R. Zayer, C. Rössl, Z. Karni, and H.-P. Seidel, *Harmonic Guidance for Surface Deformation*, Computer Graphics Forum (Eurographics 2005) **24** (2005).

[82] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum, *Large Mesh Deformation Using the Volumetric Graph Laplacian*, ACM Transactions on Graphics **24** (2005).

[83] M. Alexa, D. Cohen-Or, and D. Levin, *As-rigid-as-possible shape interpolation*, in *SIGGRAPH 2000 Conference Proceedings* (2000), pp. 157–164.

[84] R. Sumner, M. Zwicker, C. Gotsman, and J. Popović, *Mesh-Based Inverse Kinematics*, ACM Transactions on Graphics **24** (2005).

[85] P. Wesseling, *An Introduction to Multigrid Methods* (R.T. Edwards, Inc., 2004).

[86] U. Trottenberg, C. Oosterlee, and A. Schuller, *Multigrid* (Academic Press, 2000).

[87] X. Ni, M. Garland, and J. Hart, *Fair Morse Functions for Extracting the Topological Structure of a Surface Mesh*, ACM Transactions on Graphics **23** (2004).

[88] L. Grady and T. Tasdizen, *A geometric multigrid approach to solving the 2d inhomogeneous laplace equation with internal dirichlet boundary conditions*, in *International Conference on Image Processing* (2005).

[89] R. Kimmel and I. Yavneh, *An Algebraic Multigrid Approach for Image Analysis*, SIAM J. Sci. Comput. **24**, 1218 (2003).

[90] U. Clarenz, M. Griebel, M. Rumpf, M. Schweitzer, and A. Telea, *Feature sensitive multiscale editing on surfaces*, The Visual Computer **20**, 329 (2004).

[91] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, *Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid*, ACM Transactions on Graphics **22**, 917 (2003).

[92] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys, *A multigrid solver for boundary value problems using programmable graphics hardware*, in *Graphics Hardware* (2003), pp. 102–111.

[93] M. Botsch, D. Bommes, and L. Kobbelt, *Efficient linear system solvers for mesh processing*, in *Proc. of IMA conference on Mathematics of Surfaces* (2005), pp. 62–83.

[94] N. I. M. Gould, Y. Hu, and J. A. Scott, *A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations*, Tech. Rep. RAL-TR-2005-005, RAL Technical Reports (2005).

Author's Biography

Lin Shi was born on Oct. 26, 1977 in Shanghai, China. He attended Fudan University in Shanghai and earned a Bachelor of Science degree in physics in 1999. In his last semester at Fudan, Lin passed the Mini-CUSPEA examination and was granted admission to the graduate school of New York University. Two years later, Lin got his Master of Science degree in physics from NYU in 2001 and at the same time transferred to University of Illinois at Urbana-Champaign to start his Ph.D. program in computer science. Yizhou Yu became Lin's advisor in 2001 and guided Lin through his years of graduate study. During his time at UIUC, Lin has done research on various topics in computer graphics including physics-based animation and control, motion planning and optimization, texture synthesis, mesh processing and computational geometry.